# A METHOD FOR CALIBRATING A COLOR MARKING ENGINE FOR HALFTONE OPERATION

Inventor(s):

Peter A. Zuber

## IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

## A METHOD FOR CALIBRATING A COLOR MARKING ENGINE FOR HALFTONE OPERATION

## TECHNICAL FIELD OF THE INVENTION

[0001]   The present invention pertains in general to electrophotographic printers and, more particularly, to a plurality of print engines arranged in parallel to process print jobs in a parallel manner.

## CROSS REFERENCE TO RELATED APPLICATIONS

[0002]   This application is a Continuation of U.S. Patent Application Serial No. 09/923,587, filed August 7, 2001, entitled "A Method for Calibrating a Color Marking Engine for Halftone Operation," which is a Continuation of U.S. Patent No. 6,271,937, issued August 7, 2001, which is a Continuation-in-Part of U.S. Patent Application Serial No. 09/229,242, filed January 12, 1999, entitled "Color Correction of Dot Line Linearities in Multiple Print Engine System", (Atty. Docket No. TRSY-24,508) which is a Continuation-in-Part of U.S. Patent Application Serial No. 08/698,999, filed August 16, 1996, entitled "Multiple Print Engine with Virtual Job Routing", (Atty. Docket No. TRSY-23,684) which is a Continuation-in-Part of U.S. Patent Application Serial No. 08/511,641, filed August 7, 1995, entitled "Virtual Single Print Engine with Software Rip", (Atty. Docket No. TRSY-23,617) and is related to U.S. Patent Application Serial No. 08/180,636, filed January 13, 1994, and entitled "Multiple Printer Module Electrophotographic Printing Device" (Atty. Dkt. No. TRSY-22,391).

## BACKGROUND OF THE INVENTION

[0003] Electrophotographic print engines have been utilized with both printers and copiers. In a printer, the print engine is typically interfaced with a computer to select and organize fonts or bit map the images. In a copier application, the print engine is interfaced with an input device that scans the image onto the photoconductor drum of the print engine. However, a CCD device could also be utilized in this application in the form of a CCD scanner. In either of the applications, a conventional print engine for a monochrome process would typically feed a single sheet of paper and pass it by the photoconductor drum for an image transfer process and then pass it to a fuser. Thereafter, the completed sheet will be output. Multiple copy print jobs will sequentially feed the paper in a serial manner. The speed of the printer is a function of the speed at which the image can be created, the speed at which the image can be transferred to the paper and the speed of the fuser. As increased output is required, the speed of each of these elements must be increased.

[0004] In a monochrome process, only one transfer operation is required. However, in a multipass color process, multiple images must be superimposed on one another on the sheet of paper in a direct transfer system, thus requiring multiple passes of the paper or image carrier through the print engine. In a double transfer system, the image is disposed on an intermediate drum and then the composite image transferred to the paper or image carrier. In a multiple print job on a direct transfer system, this requires each sheet of paper to be printed in a serial manner by passing it through the print engine. For either the monochrome process or the color process, a conventional serial feed print engine has the output thereof defined by the speed of the input device and the speed of the print engine itself.

[0005] One technique that has been utilized to increase throughput is a tandem print engine. In a tandem print engine, multiple colors can be disposed on the sheet of paper or the image carrier at different stations that are disposed in serial

configuration. In this manner, the speed is the same for one, two, three or four color printing.

[0006] When dealing with multiple print engines, there can be a problem that exists with respect to insuring that there is adequate "color balance". In general, all color devices have a native range of colors in which they operate. This is called the color gamut of that device. Any color that falls within this gamut can be reproduced. Any color that falls outside cannot. This gamut is defined by the hardware of the device and its addressability. A monitor uses a phosphorus of some given type and is addressed in 8 bits per channel of RGB. This native gamut or range of colors changes for every different device. If it is desirable to reproduce a color on some devices, two things have to occur. First, those devices would have to be able to make that color, meaning, have that given color inside their gamuts. Second, the color would have to be correctly described, or defined as it moves from one device to another. RGB, CMYK, Lab, LCH, are all methods that devices can utilize to describe colors. They do not always have a direct translation between them, however. A method is needed to correctly translate between these methods. The analogy is as if one person would speak German and another spoke french, wherein an intermediate or interpreter would be required in order to provide communication. One method for solving this problem is to use a device independent (or color independent) space. A number of years ago, the CIE created a device independent space (XYZ) that defines color based on the light source they are viewed under, and the color response of the eye. A color independent space is a mathematical way to map device gamuts to see where they intersect. Where they intersect represents the colors they share. It is also the best platform for determining which color to use if gamuts do not intersect. Also, in this master color space, all colors are described or defined using the same terms, independent of any device. In this space, all colors are brought to a common ground. Once a color is defined in XYZ space, it can be sent and accurately reproduced on any device whose gamut in XYZ space includes that color. The reproduction of any color is accomplished by correlating the device native gamut to the color independent space.

[0007]   During a conventional print operation, toner is used up at a rate that is actually defined by the amount of information that is disposed on the given page multiplied by the number of pages.  Typically, systems incorporate some type of page counter that, when it exceeds a predetermined number of pages, indicates that the toner is low.  This, of course, is reset when a new toner cartridge is disposed in the printer.  However, this toner decision is made strictly based upon the number of pages and not the amount of toner actually depleted from the toner cartridge.  This is due to the fact that some pages have a very light toner usage compared to others.  For example, an image having a large percentage of black area associated therewith will utilize a large amount of toner, whereas a page having very light gray regions will utilize a small amount of toner.  As such, the determination of a low toner level in a cartridge is extremely inaccurate.

## SUMMARY OF THE INVENTION

[0008] The present invention disclosed and claimed herein comprises a method for calibrating the operation of a color marking engine. In one embodiment, the halftone operation in a color marking engine is calibrated wherein an image is created with a plurality of maximum density dots. A test pattern of a plurality mid-density halftone images is first created. Each image in the test pattern is associated with a different offset value of the density levels of the cyan, magenta and yellow toners associated with the color marking engine. Each of the images comprises predetermined and different combinations of the toner offsets. The test pattern is run through the marking engine to provide an output image and then a user visibly selects from the output image a desired one of the halftone images that is selected in accordance with predetermined visual criteria. Thereafter, the density level of the cyan, magenta and yellow toners is offset in accordance with the associated offsets on the selected image at the marking engine after the RIP operation.

[0009] In another aspect of the present image, a plurality of marking engines are provided with a single RIP that generates a RIPed image for distribution thereto. In one embodiment, at least one of the marking engines is subjected to the steps of running, selecting and offsetting, and, in another embodiment, all of the marking engines are subjected to the steps of running, selecting and offsetting. When more than one of the marking engines is subjected to these steps, the single test pattern is run through all of the subject marking engines to provide an output image for each subject marking engine. The user selects from each of the outputs the desired image such that each of the subject marking engines can have the offsets applied thereto after the RIPing operation.

[0010] In further embodiment of the present invention, the calibration is applied to dot linearization curves for the multiple color marking engines having a single RIP associated therewith. In order to linearize the dots for a given screen, a linearization curve is generated and applied to the RIP operation. Each linearization curve includes a plurality of offsets for each screen density value that is to be generated in

an halftone reproduction mode. The linearization curve in one embodiment is an average of the determined linearization curves for each of the marking engines. These curves are generated by running a plurality of test images therethrough, which test images correlate to a requested density value. Thereafter, the actual density is measured to provide the linearization curve for each density passed there through.

[0011] In a yet further embodiment of the present invention, the dot linearization curve is generated by averaging only linearization curves for select ones of the engines. For engines that fall outside of the bounds, the linearization curve associated therewith is excluded. For this engine, a different method is utilized to control the linearization curve. This is a mechanical adjustment wherein the parameters of the engine are actually modified whenever the average linearization curve is applied to the RIP operation.

[0012] In yet another embodiment, a method is disclosed for providing page synchronization in a color print system having a print adapter for interfacing rasterized print job data of a print job received at an input to a printer with a marking engine of the print system, comprising the steps of: processing rasterized print job data through a FIFO in the print adapter; examining the contents of the FIFO for the presence of print job data if an end-of-plane signal is detected during the processing of rasterized print job data; generating an error signal to suspend printing if the FIFO contains print job data, wherein the suspension of printing is provided to restart the print job in synchronism, upon clearing the error signal; outputting the end-of-plane signal from the print adapter with no error signal if the FIFO contained no print job data during the examining step; and repeating the preceding steps for each page until the print job is complete.

## BRIEF DESCRIPTION OF THE DRAWINGS

[0013]   For a more complete understanding of the present invention and the advantages thereof, reference is now made to the following description taken in conjunction with the accompanying Drawings in which:

FIGURE 1 illustrates an overall block diagram of the virtual printing system;

FIGURE 2 illustrates a more detailed block diagram of the virtual printing system;

FIGUREs 3a, 3b and 3c illustrate three general processing configurations;

FIGURE 4 illustrates a cutaway side view of a three module multiple print engine operated in accordance with the virtual printing system;

FIGURE 5 illustrates a flowchart illustrating the parsing operation;

FIGURE 6 illustrates a flowchart for the duplex operation for a face up output;

FIGURE 7 illustrates a flowchart for the duplex operation for a face down output.

FIGURE 8 illustrates a diagrammatic view of the stacking configuration to define a collation and gathering operation;

FIGURE 9 illustrates a block diagram of the overall job parsing operation;

FIGURE 10 illustrates the parsing operation for each printer at given jobs;

FIGURE 11 illustrates process flow for the job parsing operation;

FIGURE 12 illustrates a flowchart for the parsing operation;

FIGURE 13 illustrates a flowchart for the overall virtual job routing operation;

FIGURE 14 illustrates an overall block diagram of the system;

FIGURE 15 illustrates a block diagram of the lower level architecture between the virtual printer and the physical print engine;

FIGURES 16, 17 and 18 illustrate flowcharts for the engine allocator;

FIGURE 19 illustrates a flowchart for the resource allocator;

FIGURE 20 illustrates a prior art PCI bus structure;

FIGURE 21 illustrates a block diagram of the host adapter/print adapter of the present system;

FIGURE 22 illustrates a block diagram of the host adapter;

FIGURE 23 illustrates a block diagram of the print adapter;

FIGURE 23a illustrates a detail of the translator;

FIGURE 24 illustrates a timing diagram for the unload operation of the FIFO in the print adapter;

FIGURE 25 illustrates a block diagram of the manner in which the device profiles are handled in a prior art system;

FIGURE 26 illustrates a block diagram of a conventional system for balancing color space;

FIGURE 27 illustrates the color balancing operation of the present invention;

FIGURE 28 illustrates a flowchart depicting the calibration method;

FIGURE 28a illustrates a test pattern for adjusting the density levels on each machine;

FIGURE 28b illustrates a flowchart for adjusting the density levels of the individual toner levels on each machine;

FIGURE 28c illustrates a flowchart depicting the adjustment operation;

FIGURE 29 illustrates a test pattern for the fine tune operation for bi-level and quad-level;

FIGURE 29a illustrates an alternate embodiment of the patch of FIGURE 29;

FIGURE 30 illustrates a flowchart for the fine tune operation;

FIGURE 31 illustrates a plot of a dot linearization curve for a halftone print operation;

FIGURE 32 illustrates a diagrammatic block diagram of the correction operation for the RIP with halftone correction;

FIGURE 33 illustrates a flowchart depicting the method for generating the average dot linearization curve;

FIGURE 34 illustrates a flowchart depicting the RIP operation with correction provided by the averaged dot linearization;

FIGURE 35 illustrates a block diagram of the RIP;

FIGURE 36 illustrates a flowchart for the output plug-in portion of the RIP;

FIGURE 37 illustrates a flowchart for the toner calculation operation;

FIGURE 38 illustrates a flowchart for the power-on sequence operation;

FIGURE 39 illustrates a flowchart for the merge operation;

FIGURE 40 illustrates a flowchart for the stack control operation;

FIGUREs 41 and 42 illustrate flowcharts for the page synchronization operation;

FIGURE 43 illustrates a block diagram of an embodiment of the present invention utilizing an automatic finishing step; and

FIGURE 44 illustrates a flowchart for the embodiment of FIGURE 43.

## DETAILED DESCRIPTION OF THE INVENTION

[0014]    Referring now to FIGURE 1, there is illustrated a block diagram of the overall operation of the virtual printing system.  A plurality of workstations 10 are provided, which workstations 10 comprise general personal computers or other terminals that allow a user to create print jobs.  Each of the workstations is networked through a network interface 12, which is a conventional type of general network interface such as an Ethernet® network interface.  This allows each workstation 10 to send its print job to a central processor 14, which processor is operable to process the print jobs in accordance with the system of the present invention and distribute these print jobs to multiple print engines 16.  As will be described hereinbelow, the processor 14 is operable to disassemble the print job, parse the print job into different pages and distribute the parsed pages in a predetermined manner in accordance with the present invention.  It should be understood that a print job, although initiated as a series of pages, is sent as a single job to a printer.  Typically, printers receive the print job in a conventional manner, which is a string of digits and the printers determine whether the codes are for an end of page command, etc.  However, most print operations within a given workstation 10 are designed such that the print job is to be sent to a single printer and, therefore, the codes are all "bundled" in a common string or job.  As will be described hereinbelow, in order for the pages to be parsed, it is important to first determine what the beginning and the end of a print job is, then determine what printer to send that distinct and separate page to, in accordance with the system of the present invention.

[0015]    Referring now to FIGURE 2, there is illustrated a more detailed block diagram of the operation of the processor and the parsing operation for distributing the parsed pages to the various print engines 16.  The job is received in a serial manner, and is "spooled" in a print spooler 20.  This is then passed to a software RIP engine 22 which is operable to essentially decode the print string that is received from the print spooler 20.  This effectively divides each print job into pages.  These pages are then stored in page buffers 24.  Each page in the page buffer essentially

constitutes a single print job, such that any print job received from the workstations 10 will then be parsed into a multiple print job file. For example, if a thirty page document were to be sent, this would be sent as a single print job, which would be encoded as such. The software RIP engine 22 is then operable to divide this into thirty separate print jobs.

[0016] Once the pages are stored in the page buffer 24, then the pages are sent to an image task manager 26 to determine how to organize the pages. This operates in conjunction with an engine manager 28 to determine which of the print engines 16 the job is to be passed to. In order to effectively increase the throughput from the engine manager 28, there are provided interface circuits 32 which are referred to as Peripheral Connect Interface (PCI) adaptors. Each print engine 16 has a PCI 32 associated therewith. Therefore, the engine manager 28 interfaces with the PCIs 32 through a parallel bus 36, such that data can be transferred thereto at a fairly high data rate, which is the bus transfer data rate of the processor 14. The PCIs 32 therefore provide an increased rate of transfer to the print engine 16. The print engines 16 then place their output into a separate output bin 40 for each of the print engines 16.

[0017] As will be described hereinbelow, the image task manager 26 is operable to arrange the copies such that they can be placed in the output bins 40 in a predetermined order. For example, if there were two print engines, each with a 100 sheet paper supply and four print jobs of 50 copies each were to be sent to the printers and the workstation 10, the system of the present invention would parse these print jobs such that the first two print jobs went to the first print engine and the second two print jobs went to the second print engine. If, alternatively, the two print engines with the one hundred sheet paper supplies handled two print jobs, one at a 150 sheets and one at 50 sheets, then the first print engine would receive the first 100 sheets from the first print job, the second print engine would receive the first 50 sheets of the first print job and the second 50 sheets of the second print job. However, they would be sent to the printer in such a manner that when the paper output trays were unloaded and stacked together, the jobs would be arranged in the

appropriate manner. Therefore, even though there are multiple printers, to the user they appear as a virtual single printer. All decision making is made in the processor 14.

[0018] Referring now to FIGUREs 3a-3c, there are illustrated the various configurations illustrating the transfer of data between an input and a print engine. In FIGURE 3a, there is illustrated a general diagram of a software RIP processor 42, which is operable to generate the data necessary to transfer to a print engine 46. However, this is effected over a conventional parallel port 48. In this configuration, the software RIP processor 42 is relatively fast, whereas the print engine 46 is relatively slow. Of the time to print, three percent of that time is occupied by the operation of print engine 46, seventy percent is occupied by the software RIP processor 42 and twenty-seven percent is occupied by transferring the data from the processor 42 to the print engine 46. Therefore, the parallel port 48 becomes a key factor in the printing time. In FIGURE 3b, software RIP processor 42 is connected to the print engine 16 via a PCI 50. In this configuration, ninety-five percent of the print time is occupied by the software RIP processor 42, three percent by the print engine 16 and five percent by the PCI 50. Therefore, by reducing the transfer time from the processor 42 to the print engine 16, an increase in speed has been seen. In FIGURE 3c, there is illustrated a fairly conventional system wherein a processor 52 is provided, which can be a conventional PC for assembling the print job in a conventional manner and transferring it via a parallel port 54 to an engine 58, which is a conventional print engine having an internal RIP 60 associated with a marking engine 62. The processor 52 is relatively fast, and it occupies virtually no time. Seventeen percent of the print time is taken passing the data to the RIP 60 through the parallel port 54, whereas eighty percent of the print time is occupied with the RIP 60 and only three percent by the marking engine 62.

[0019] Referring now to FIGURE 4, there is illustrated a cutaway side view of a three print engine module parallel printer which includes three print engines 136, 138 and 40, all stacked one on top of the other. Each of the engines 136-140 is a multi-pass engine and includes a transfer drum 142 and a photoconductor drum 144.

The photoconductor drum 144 rotates in a counterclockwise direction and is pressed against the transfer drum 142 to form a nip 146 therebetween. The photoconductor drum 144 is operable to have the surface thereof charged with a corona 148 and then an imaging device 150 is provided for generating a latent image on the charged surface of the photoconductor drum 144. The undeveloped latent image is then passed by four developing stations, three color developing stations, 152, 154 and 156 for the colors yellow, magenta and cyan, and a black and white developing station 158. The color developing stations 152, 154 and 156 each have a respective toner cartridge 160, 162 and 164 associated therewith. The black and white developing station 158 has a black and white toner cartridge 166 associated therewith. Although not described hereinbelow, each of the developing stations 152-168 and toner cartridges 160-166 can be removed as individual modules for maintenance thereof.

[0020] During the print operation, the photoconductor drum 144 is rotated and the surface thereof charged by the corona 148. An undeveloped latent image is then formed on the surface of the photoconductor drum 144 and then passed under the developing stations 150-158. In a multi-pass operation, the latent image is generated and only one color at a time utilized in the developing process for the latent image. This latent image is then passed through the nip 146 and transferred to an image carrier, such as paper, which is disposed on the surface of the transfer drum 142. Thereafter, the surface of the drum 144 is passed under a cleaning station 168, which is operable to remove any excess toner particles which were not passed over to the transfer drum 142 during the transfer operation and also discharges the surface of the drum 144. The system then begins generation of another latent image, either for a different color on the same sheet of paper or the first color on a different sheet of paper.

[0021] In the color operation, multiple passes must be made such that the image carrier, i.e., paper, remains on the surface of the transfer drum 142 for the multiple passes. In the first pass, the first latent image is transferred to the surface of the transfer image carrier and then the image carrier maintained on the transfer drum 142. The next latent image of the next color is superimposed on the first latent

image, it being noted that the registration is important. This registration is provided by the mechanical alignment of the various drums, drive mechanisms, etc. Thereafter, the third color latent image is disposed on the image carrier followed by the fourth color latent image.

[0022] After the last color latent image is disposed on the image carrier in the color process, a picker mechanism 172 comes down on the surface of the transfer drum 142 in order to lift up the edge of the image carrier or paper. This is then fed to a fuser mechanism 174.

[0023] The image carrier is typically comprised of a predetermined weight paper. The transfer drum 142 utilizes electrostatic gripping for the purpose of adhering the paper to the surface of the transfer drum 142 for multiple passes. This therefore utilizes some type of charging mechanism for charging the surface of the drum 142 at an attachment point 176 where the paper is fed onto the surface of the transfer drum 142. The transfer drum 142 is, in the preferred embodiment, manufactured from a controlled resistivity type material that is disposed over an aluminum support layer which is a hollow cylindrical member. A voltage supply is provided that provides a uniform application of voltage from the voltage supply to the underside of the resilient layer that is disposed over the surface of the aluminum support member. This resilient layer is fabricated from a carbon filled elastomer or material such as butadaiene acrylonitorile, which has a thickness of approximately 3 mm. Overlying this resilient layer is a controlled resistivity layer which is composed of a thin dielectric layer of material at a thickness of between 50 and 100 microns. This controlled resistivity layer has a non-linear relationship between the discharge (or relaxation) point tying and the applied voltage such that, as the voltage increases, the discharge time changes as a function thereof. The paper is then disposed over the surface of the drum. The construction of this drum is described in U.S. Patent No. 5,459,560, issued October 17, 1995, and entitled, "Buried Electrode Drum for an Electrophotographic Print Engine with a Controlled Resistivity Layer", which is a continuation-in-part of U.S. Patent No. 5,276,490, and entitled, "Buried Electrode

Drum for an Electrophotographic Print Engine", which U.S. Patents are incorporated herein by reference.

[0024]    The paper is retrieved from one of two paper supply bins 178 or 180. The paper supply bin 178 contains one type of paper, typically 8½" x 11" paper, and the paper bin 180 contains another type of paper, typically 8½" x 14" paper. The paper bin 178 has the paper stored therein selected by a first gripping roller 182, which is then fed along a paper path 180 into a nip 182 between two rollers and then to a nip 184 between two rollers. This is then fed to a paper path 186 to feed into a nip 188 between two rollers. The paper in the nip 188 is then fed into a nip formed between two precurl rollers 190 and 192, which have different durometers to cause the paper to have a curl bias applied thereto in the direction of the curvature of rotation of the transfer drum 142. The operation of the pre-curl rollers is described in detail in U.S. Patent No. 5,398,107, issued March 14, 1995, and entitled, "Apparatus for Biasing the Curvature of an Image Carrier on a Transfer Drum" (Atty. Dkt. No. TRSY-22,574). The paper from the bin 180 is extracted by a gripping roller 189 and pushed along a paper path 191 to the nip 188 and therefrom to the pre-curl rollers 190 and 192.

[0025]    The paper is fed from the nip between the two pre-curl rollers 190 and 192 at the attachment point 176. At the attachment point 176, an attachment electrode roller 194 is provided which is operable to operate on a cam mechanism (not shown) to urge the roller 194 against the surface of the drum 142 to form the attachment nip 176. This is done during the initial attachment of the paper to the drum 142. Typically, this attachment electrode roller 194 is connected to ground. The surface of the drum 142 is charged to a positive voltage of between 800-1,000 volts. The voltage is disposed on the surface of the drum 142 by a positive electrode roller 196 that contacts the surface of the drum 142 at a point proximate to the photoconductor drum 144. Since the electrode 194 is grounded, the voltage will decrease along the surface thereof until a lower voltage is present at the attachment point 176. When the paper reaches the transfer nip 146, the portion of the surface of the photoconductor drum 144 in the nip 146 has a potential thereof reduced to ground

such that the charged particles will be attracted from the surface of the photoconductor drum 144 to the surface of the paper on the drum 142.

[0026]    For a multiple pass operation, the attachment electrode 176 will be pulled outward from the drum and the paper allowed to remain on the drum and go through the transfer nip 146 for another pass.  When the final pass has been achieved at the transfer nip 146, the picker 172 is swung down onto the surface of the drum 142 to direct the paper on the surface of the drum 142 to the fuser 174.  A discharge electrode 198 is then swung down into contact with the drum 142 to provide a discharge operation before the surface of the drum enters the nip 176 for the next paper attachment process.

[0027]    When the paper is fed into the fuser 174, it is passed into a nip between two rollers 200 and 202, both of which have different durometers.  Typically, there is one roller that is formed from a metallic material and one roller that is formed of a soft material.  The rollers are oriented with the roller 200 having the smaller durometer, such that a reverse bias curl will be applied to the paper that is the opposite direction of the curvature of the drum 142.  This will remove the curvature added to the paper. One of the rollers 200 is heated such that the transferred image is "fused".  The paper is then fed into a paper path 204 by a pair of rollers 206.  The paper path 204 is fed to a set of output rollers 208, which feed bins 210, 212 and 214 for each of the printers  136, 138 and 140.  Again, these are conventional print engines, although the speeds of the print engines may be different.

[0028]    Referring now to FIGURE 5, there is illustrated a flowchart depicting the operation of the virtual printing system.  For this description, the following terms are defined:

    N = number of pages in a single document
    M = copies
    E = number of engines
    P = number of pages
    I = the engine number.

[0029] The flowchart is initiated at a start block 230 and then proceeds to a decision block 232. A decision block 232 multiples the number of pages N by the number of copies M and determines whether this number if greater than or equal to the number of engines. If not, then the program flows along a "N" path to a function block 234 to utilize only a single engine for the print job. However, if the number is greater than the number of engines, then the program proceeds along the "Y" path to a decision block 236 to determine the number of copies M is greater than the number of engines E. If not, the program flows along a path "N" to a decision block 238 to determine if the number of pages in a single document "N" is greater than or equal to the number of engines. If not, the program will flow along a "N" path to a function block 240 to utilize the only M engines with the $I^{th}$ copy in the $I^{th}$ engine. Therefore, if there are ten engines and only five copies, then the fifth copy of a job will be in this fifth engine. If, however, the number of copies in a single document is greater than the number of engines, then the program will flow along a "Y" path to a function block 242 wherein the copies will be distributed in accordance with the equation:

$$P = \frac{N \times M}{E} \tag{1}$$

[0030] If it was determined in the decision block 236 that the number of copies M was greater than the number of engines with the number of copies times the number of pages in a single document also being greater than the number of engines, then the program flows along the "Y" path from decision block 236 to a decision block 244 to distribute copies. These are distributed in accordance with the algorithms illustrated in FIGURE 5 with respect to four of the engines $E_1$, $E_2$, $E_3$ and $E_4$. $E_1$, $E_2$ and $E_3$ are also associated with function blocks 246, 248 and 250, each operating in accordance with the above equation, one associated with function block 242. However, $E_4$ will flow to a function block 256 wherein the distribution will be as follows:

$$P_4 = N \times M - (P_1 + P_2 + P_3) \tag{2}$$

[0031]  Referring now to FIGURE 6, there is illustrated a flowchart depicting the operation for a duplex print job. In the flowchart of FIGURE 6, a face up output is considered which is initiated at a block 260. The function block then flows to a decision block 262 to determine if the value of N is even. If so, the program flows to a function block 264 to print the jobs –2, –4 ..., 2. The program then flows to a decision block 266, which determines whether the value of N is odd. However, if N was odd at decision block 266, the program would flow along the "N" path to the output of the decision block 266 and then to a function block 268 to print the N + 1 copies and blank copies and then print the N - 1, N - 3, ... 1 pages. The flowchart would then flow to a function block 270. It is noted that if N is even at decision block 266, the program would flow to the function block 270. Function block 270 is a function block wherein a user annually turns the output stack 180° without flipping the stack and then puts it back in the drawer of the printer from which it came. The program then flows to a decision block 74 to determine if the value of N is even, and if so, to the function block 270 along the "Y" path to print the pages 1, 3, 5, ... –1, and then to a decision block 278 to determine if the value of N is odd. The program at this point will flow along the "N" path to a N block 280. However, if the value of N is determined to be odd at decision block 274, the program will flow through the output of decision block 278 and to the input of a function block 282 which will print the pages 1, 3, 5, ... N.

[0032]  Referring now to FIGURE 7, there is illustrated a flowchart depicting the duplex operation with a face down output, which is initiated at a block 284 and then proceeds to a decision block 286 to determine if the value of N is even. If so, the program then flows to a function block 288 along the "Y" path to print the pages 2, 4, 6, ... N. If it was determined that the value of N is odd, the program would flow along an "N" path to a function block 290 to print the pages 2, 4, 6, ... –1. The program 288 would flow to a decision block 294, which determines if N is odd and, if not, flows along a "N" path to the output of function block 290, the output of a decision block 294 is input to function block 290. The output of function block 290 flows through a function block 296, as well as the output along the "N" path of

decision block 294. Decision block 296 indicates the manual operation wherein the user flips the output stack without turning it 180° and then inputs it back into the drawer of the printer from which it was obtained. The program will then flow to a decision block 298 to determine if the value of N is even. If so, the program flows along a "Y" path to a function block 300 and the pages 1, 3, 5, ... –1 and then to the input of a decision block 302. If the value of N is odd, the program flows along the "N" path from decision block 298 to the output of decision block 308 and to a function block 306 to print the pages 1, 3, 5, ... N. The output of the decision block 302 along the "Y" path also flows to the function block 306 when N is even, and the flowchart flows along the "N" path to an "END" block 310, this being the path from the function block 306.

[0033]    In general, to provide routing of the different images or pages to the various print engines 16 provides the ability for the system to make certain decisions about how a particular job is output. This is facilitated by the fact that each print job, when it has been initially assembled and transmitted to the system, is disassembled during the RIP operation and then buffered in the form of separate and distinct pages. With knowledge of print related information for each of the pages in a given job, additional processing can be performed on those pages. This processing can be in the form of routing the pages to engines that are more adapted to the particular printing operation associated with that particular page. For example, a page that has no color on it would be better handled by a dedicated black and white engine as opposed to a page having color on it being handled by a color engine. Typically, color engines, although they do have a black and white mode, operate best in the color mode. Dedicated black and white engines are significantly faster than color engines operating in the black and white mode.

[0034]    One example of a type of problem that occurs when attempting to handle a print job having mixed color and black and white sheets is that where a high speed black and white print engine can be provided to print black and white pages more efficiently than a color engine with black and white capability. By incorporating different types of engines as a portion of the overall virtual system in the print engine

16, a higher level of versatility will be facilitated. One reason that color engines have difficulty in switching from color to black and white is that these types of engines have an internal sequencer that must be programmed when changing printing mode (color versus black and white). To reload the sequencer requires the engine to wait for the current page to exit the fuser.

[0035] In order to facilitate this system, the print engines 16 are grouped into physical engines of similar characteristics. For example, a set of four color physical engines can be configured as one virtual engine and a set of four high speed black and white engines can be configured as another virtual engine. To the outside world, these virtual engines simply appear as a high speed entity (the speed is equal to the sum of the individual engines rated print speed). In this case, the outside world will see two high speed devices, one of which has color capability.

[0036] There are two methods of collation that exist in the present system, automatic and gather. In an automatic collation system, collated documents are generated and the user is then required to stack them and perform whatever finishing is required. For example, if there were 10 copies of a 6 page document, the following sequence would be printed:

1, 2, 3, 4, 5, 6

1, 2, 3, 4, 5, 6

•

•

1, 2, 3, 4, 5, 6 (10th copy)

This particular sequence will result in the collated copies being present in the output bin. Of course, the particular number of engines can be configured and controlled to determine how to most efficiently provide the collated copies. However, once the copies are in the stack, they are in a sequential collated configuration.

[0037] The gather method of collation is employed when external finishing and collation is available. In this case, all copies of each individual page must be printed before the next page in the document is printed. Therefore, without the multiple

engine concept as described hereinabove, this would require one engine to perform all of the necessary copies of page 1, then the necessary copies of page 2, etc. This would result in a stack that had M sheets of page 1, followed by M sheets of page 2, etc. In the above example with the 10 copies of the 6 page document, the following sequence would be present:

$$1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1$$
$$2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2$$
$$3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3$$
$$4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4$$
$$5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5$$
$$6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6$$

This arrangement, of course, would result in a single stack of the 6 pages, which would then be placed in the external finishing and collating hardware.

[0038]    Referring now to FIGURE 8, there is illustrated a diagrammatic view of the collation operation and the gather operation. In this particular example, there is illustrated an example wherein a document having "N" pages and "M" copies is to be provided for. In a collation operation, it can be seen that the final stack results in a sequence of the first page, the second page, the third page and the Nth page followed by the first page of the next copy, the second page, third of the Mth copy. This will continue for all M copies. In the gather method, the stack is configured of M of the first pages, M of the second pages and M of the third pages, only 3 pages illustrated for simplicity purposes. This will comprise a single stack.

[0039]    There are instances where the gather method of collation has a significant speed advantage over the automatic collation method. Using the above noted example with 6 pages and 10 copies, assume that pages 2, 4 and 6 are color pages and pages 1, 3 and 5 are black and white pages. Also assume that it is desired to have 100 copies in order to notice a difference between the two operations. If the print job is to be printed with the color virtual engine, a time penalty is allotted each time the color engine is switched between the color mode and the black and white

mode and back to the color mode. In one type of color engine, a Canon P320 model, this penalty is on the order of 8 seconds. In this example, the time penalty will occur between pages as follows:

1-2 black and white to color switch

2-3 color to black and white switch

4-5 black and white to color switch

5-1 color to black and white switch

This will result in 32 seconds for each copy. If this is multiplied times 100 copies, it will result in almost an hour of lost time just for switching between the color and black and white modes on this engine. With a four engine virtual engine, the print distribution would look as follows:

**Table 1**

| Engine 1 | Engine 2 | Engine 3 | Engine 4 |
|----------|----------|----------|----------|
| 1-6 x 25 | 1-6 x 25 | 1-6 x 25 | 1-6 x 25 |

It can be seen from Table 1 that each physical virtual engine in the four engine virtual engine would print 25 copies of the 6 page document in the order 1, 2, 3, 4, 5, 6, such that each physical engine would have a time penalty of 25x8=200 seconds.

[0040]   If the gather collation method is utilized in the above example, the pages would be divided among the engines as set forth in Table 2:

**Table 2**

| Engine 1 | Engine 2 | Engine 3 | Engine 4 |
|----------|----------|----------|----------|
| 1 x 100  | 2 x 50   | 4 x 100  | 5 x 50   |
| 2 x 50   | 3 x 100  | 5 x 50   | 6 x 100  |

In this example of Table 2, the first engine prints 100 copies of page 1 and 50 copies of page 2, engine 2 prints the other 50 copies of page 2 and 100 copies of page 3 and so on. This results in each engine only making one mode change in order to allot

only a single 8 seconds per engine and, since they are running in parallel, it is only 8 seconds for the entire virtual engine.

[0041] A further optimization could be applied to achieve the following print sequence set forth in Table 3:

**Table 3**

| Engine 1 | Engine 2 | Engine 3 | Engine 4 |
|----------|----------|----------|----------|
| 1 x 100  | 3 x 50   | 2 x 100  | 4 x 50   |
| 3 x 50   | 5 x 100  | 4 x 50   | 6 x 100  |

In Table 3, the black and white pages have been grouped and printed on engines 1 and 2 and the color pages have been grouped and printed on engines 3 and 4. As such, there are no penalty hits due to mode changes, since each engine is never required to change modes. In order to achieve this configuration, of course, it is necessary to know whether or not a page contains color information. This information is determined after the RIP operation in the RIP 22, as described hereinabove.

[0042] Referring now to FIGURE 9, there is illustrated a block diagram illustrating the process flow for both the automatic collation operation and the gathering collation operation. A document is input to the system, as represented by a box 320, which in this example is a three page document where the value of "N" is equal to three. The next operation will be defining the job, which is represented by a box 322 wherein the number of copies "M" is defined. Thereafter, it is necessary to define the manner in which the stack will be configured. This stack is basically the overall result at the output of the engine. Since this is a virtual print engine, multiple output bins will be utilized to form the stack. Of course, if there were a single engine, the stack would be that which results in the output bin of the single engine. With multiple engines, it is only necessary to extract the documents from each of the engines in sequence and place them into a single stack such that the overall stack will be as if it were printed with one engine. This is defined as the image task manager (26) in FIGURE 2. The operation of the stack definition will be determined

to be an automatic collation or gathering collation operation. If it is automatic, the process will flow to a block 326 to place the individual pages in the correct order and then output them to a parser 328 which then distributes them to a plurality of print engines 330.

[0043]   If the gather operation were selected at the stack define block 324, the stack would be defined as set forth in a block 328 which would be as described above with respect to FIGURE 8, i.e., the stack would be M copies of page 1, followed by M pages of page 2 and M pages of page 3. This would be input to a parser 332 for output to a plurality of print engines 334 which are configured as a single virtual engine. By configuring them as a single virtual engine, the output bins from the print engines 334 will in fact provide the stack as defined in the box 328.

[0044]   For the gathering operation, in order to determine how to define the parsing operation for the stack, it is necessary first to determine the number of sheets that will be present for each job. This is the number of pages in a document "N" multiplied by the number of copies "M". It is then necessary to determine how many sheets are to be accommodated by each engine which number of sheets is the value of "Q" which is defined by the following equation:

$$Q = \frac{NxM}{E}$$

Where: M = copies

　　　　N = number of pages in the document

　　　　E = number of engines

　　　　Q = number of pages per engine in gather operation

This is illustrated diagrammatically in FIGURE 10 where it can be seen that the boundary for each of the print engines in the overall virtual printer (there being illustrated 4 printers) is defined such that, when finished, the outputs from each of the printers can be sequentially stacked to form the stack defined in block 328. The boundary for each of the "Q" values is something that is defined, in the present

example, by equally dividing the total number of pages by the number of printers. However, the boundary could be defined as the page break. If there were 4 printers and 4 or less pages, it would be quite easy to route each page to a separate printer. It is only necessary to fix this boundary such that the right engine will get the right page. For example, if one of the printers in the set of printers defined as the virtual printer were a color printer and one page were color, the system could route the color page to that particular printer. It is then only necessary for the operator to understand that this particular configuration requires the outputs to be assembled in a predetermined manner.

[0045] Referring now to FIGURE 11, there is illustrated a diagrammatic view of the overall parsing operation which is defined with a parsing block 340. This parsing block is operable to determine how the job is to be split up between the engine and this is then output in the appropriate order and stored in a page buffer 342. The page buffer then sequentially outputs the data to a block 344, which basically performs the operation of the engine manager, that is, it routes the copies to the correct print engine. The entire operation is facilitated in a manner that allows pages to be output by the parsing block 340 in a particular sequence that will match that of the engine selection. Each of the engines may not print at the same speed and, therefore, the sequence that the pages are output by the parsing block 340 may not be the exact sequence that they are input to the print engines. The block 344 determines which pages goes to which engine and operates in conjunction with the parsing operation to provide the overall desired result. For example, one printer may be a much higher speed printer whereas the second printer may deal with a higher resolution page. Since the higher resolution pages take longer to print, it may be desirable to output a number of pages to the higher speed printer such that it completes its portion of the job prior to the lower speed printer. This will result in the parsing block 340 outputting the black and white pages initially in order to maintain the speed rate for the high speed printer and at a slower rate color pages for the low speed printer, such that they will be fed to the low speed printer at the appropriate rate. This allows a single page buffer 342 to be provided for all of the printers.

[0046] In another aspect of the present invention, the system is operable to divide a particular job into multiple jobs and provide "virtual job routing". When virtual job routing is utilized, the job is examined and then converted into two separate jobs, depending upon whether it may be faster to group the operations, for a job such as a mixed color and black and white job. The color job would then be defined as a single group and would be submitted to a color virtual engine, wherein the black and white job would be grouped separately and submitted to a high speed virtual engine. Each of the color virtual engines and black and white virtual engines are a combination of multiple engines. As an example, a high speed black and white engine is provided that is assumed to be four times faster than the black and white mode of the color engines. In the virtual job router, two jobs would be generated as illustrated in Tables 4 and 5.

### Table 4 - Black and White Job

| Engine 1 | Engine 2 | Engine 3 | Engine 4 |
|----------|----------|----------|----------|
| 1 x 75   | 1 x 25   | 3 x 50   | 5 x 75   |
|          | 3 x 50   | 5 x 25   |          |

### Table 5 - Color Job

| Engine 1 | Engine 2 | Engine 3 | Engine 4 |
|----------|----------|----------|----------|
| 2 x 75   | 2 x 25   | 4 x 50   | 6 x 75   |
|          | 4 x 50   | 6 x 25   |          |

In the example noted above with respect to Tables 4 and 5, it can be seen that the basic job has been divided into two completely separate jobs that will then be submitted to a job manager, with each to be completed by its respective virtual engine. This involves both error handling and other aspects that would be handled by a single printer. It can be seen from the example from Tables 4 and 5 that a significant speed advantage has been achieved by not burdening the color engines with the black and white pages. In many cases, a cost savings will also be achieved since it is cheaper to print black and white pages on a black and white engine than it

is to print on a color engine. Of course, in this example, the number of color pages was equal to the number of black and white pages. In most instances, this is not the case, with the black and white pages usually outnumbering the color pages, which makes the virtual routing a more important process.

[0047]  Referring now to FIGURE 12, there is illustrated a graphical representation of the process. The process is initiated at the software RIP in a block 350, which is operable to retrieve the initial multi-page document and the RIP the document into separate pages, which pages are separate and distinct and have associated therewith parameters that define the nature of the document as to printing, i.e., whether it is color or black and white, the possible resolution of it, bit depth thereof, etc. The process will then flow to a block 352, wherein the job will be defined as being a virtual job routing job and will be divided into two or more jobs. In the present example, there is a black and white job and a color job. The process will then flow to a virtual job router block 354, which is the parsing operation. The black and white job is routed to a first job block 356 and the color job is routed to a second job block 358. Both of these jobs are handled by a job manager 360, illustrated in phantom. The job manager will route the black and white job to a first virtual engine, represented by a block 362, which has associated therewith four black and white print engines 364. The job manager will route the second job associated with the block 358 to a second virtual engine 366, having associated therewith four color print engines 368. It should be noted that the job manager 360 will essentially perform the operation of the parsing and will ensure that pages that are extracted from the internal page buffer (not shown) will be routed to the appropriate engine in the appropriate manner and at the appropriate time. It is noteworthy that the pages will be routed to the virtual engine 362 at a much higher rate and they will be routed to the virtual engine 366, as the color engines are typically slower than the high speed black and white engines. It is only necessary that the integrity of the overall stack that was defined in the stack block 328 of FIGURE 9 be maintained. It is important to note that the print engines 334 in FIGURE 9 basically will be grouped as the two virtual engines 362 and 366 when virtual job routing is utilized and the gather collation method is utilized. In general, virtual job routing will utilize the

gather collation method. It should be understood, however, that the primary advantage provided by virtual job routing is that a particular page can have the parameters thereof examined after the page has been assembled separate from the initial multi-page print job, and a determination made as to how to handle that particular job. This will allow the job to be routed to the most efficient engine.

[0048] Referring now to FIGURE 13, there is illustrated a flow chart for the overall virtual job routing operation. The flowchart is initiated at a start block 370 and then proceeds to a block 372 in order to perform the software RIP operation. This software RIP operation is the operation described hereinabove that allows the image for each page to be extracted out of the original input print job from the workstation to provide a stand alone page. After the document has been ripped and stored as a single page, the program then flows to a function block 374 to receive an input as to the number of pages in the document "N". The program then proceeds to a function block 376 to receive the number of copies that are to be made of the document, this being the value "M". The program then flows to a decision block 378 to determine if the pages are determined to have job specific pages, i.e., certain pages are color and certain pages are black and white, or, alternatively, that certain pages require processing by a certain one of the print engines or a certain one of the group of print engines as a virtual job. If so, the program will flow along the "Y" path to a function block 380 in order to parse the jobs. The program will flow to a function block 382 to define the "Q" boundary for the jobs, i.e., how many sheets are to be routed to each engine, and then to a function block 384. Function block 384 creates the stack for each job with the defined "Q" boundaries disposed therein, which "Q" boundaries defined which portion of the stack goes to which printer. The program will then flow to a function block 386 to transfer the pages to the page buffer and then to a function block 388 in order to transfer the pages to the particular printer of the function of the job that is being performed and the printer that is designated. The program will then flow to an end block 390.

[0049] If the decision block 378 had determined that there were no job specific pages, the program will then flow along the "N" path and proceed with the normal

virtual print engine parsing, as described hereinabove, this indicated in a function block 392. The program will then flow to a function block 394 to transfer the pages to the page buffer as a function of the parsing operation and then to a function block 396 to transfer the pages from the page buffer to the virtual printer. The program will then flow to the end block 390.

[0050] Referring now to FIGURE 14, there is illustrated a general block diagram of the overall system illustrating in more detail the operation of the electronic collator and the Print Station Manager. As described above, whenever a document is received, it is processed through a software RIP, such that individual bit map pages can be defined, which individual bit mat pages constitute images, which images are independent as to the print parameters associated therewith, such as color, bit resolution, bit depth, etc., but are associated with a document. In the block diagram of FIGURE 14, a multi-page document is input to the system. This document is input from a PC or some type of user. In general, each of the multi-page documents constitutes a job. In FIGURE 14, there are illustrated four documents, a multi-page document 400, a color document 402, a hybrid document 404, and a black and white document 406. The multi-page document 400 may be any type of document which includes multiple pages. Document 402 is a document that is defined as operable to be printed on a color print engine. The document 404 is a hybrid document which can have black and white pages and color pages. The document 406 is in an entirely black and white document. It should be understood that the virtual print engine of the present invention is operable to convert each of the documents into single individual bit-mapped pages, which images are stored on a page-by-page basis for each document. Thereafter, as described above, these pages are distributed to various engines in a parallel manner, depending upon the characteristics of the page, the availability of the engine and, in general, how best to match a given page in accordance with its characteristics with a given engine in accordance with that engine's characteristics. This facilitates maximum throughput through the engines and maximizes the use of the engines' capabilities.

[0051]   There are illustrated four print engines 408, labeled PE1, PE2, PE3 and PE4.  In a preferred embodiment, the print engines 408 are realized with a Canon P320 color laser printer print engines.  Each engine is rated at 3 ppm for four-color pages and 12 ppm for black pages.  In a mixed document, black-only pages are printed at 12 ppm and color pages at 3 ppm.  The print engines are configured to accept letter-size (or A4) or legal-size paper and prints on only one side, although double-sided printing could be facilitated with another type of engine.  In general, the print engine in the preferred embodiment utilizes a 60-to-80 micron spot and images at 600 x 600 dpi.

[0052]   The print engines 408 are interfaced on the input side thereof with a print station manager 410 through an interconnect 412, the interconnect 412 defined hereinabove as the PCI interface, which will be described in more detail hereinbelow.  The print station manager 412 in general is operable to define the way in which jobs are initially assembled and reported to the printers.  The output of each of the print engines 408 is basically disposed in bins associated with the print engines and these are then fed to a document assembly operation, represented by a block 414.  This is any type of technique for retrieving documents from the print engine and placing them in an appropriate order.  In one embodiment, this is done manually.

[0053]   The Print Station Manager 410 receives the input therefrom from an electronic collator 418, which is in part described hereinabove with reference to FIGURE 9 as the stack define block 324.  This electronic collator 418 receives an input from a job parser 412, which is operable to retrieve pages from a memory 414, which pages are basically compressed bit maps.  These compressed bit maps are oriented such that each page defines a bit mapped image, with each page having associated therewith information regarding the parameters of the page with respect to printing.  These parameters define such things as resolution, bit depth, color/ black and white, etc.  The compressed bit maps, as described above, are derived from a software RIP 416, which interfaces with a spooler/OPI server 419.  The spooler 419 is operable to receive the documents 400-406.

[0054] The software RIP 416, as described hereinabove, is a PostScript RIP created by *Harlequin* as the *Level 2 ScriptWorks* software RIP running under Windows NetWork. The configuration for this software RIP in the present embodiment requires 64 MegaBits of memory and a 4-GB hard disk for storing compressed rasterized pages. In general, the software RIP can rasterize an entire job, store it on disk, and then start sending it to another recording engine, this being the preferred mode when utilizing slower recording engines and when rasterized data must be saved on disk for reuse later. Alternatively, the RIP is operable to pass rasterized pages to the print engines at the same time it writes them to disk. This is what is referred to as the "writethrough" mode, as it effectively results in bypassing of the print queue. In cases where the engines are not able to keep up with the RIP, the system terminates transmission of data directly to the engine and continues to pass pages to the disk, from which they are fed to the engine at a later time. The bit maps are compressed utilizing a compression algorithm which is referred to as a "pack bit" compression algorithm, a conventional algorithm. These rasterized pages are available to be printed and reprinted, either on a document or a page basis, at any time.

[0055] This software RIP is operable to provide dispersed screening techniques and conventional halftone screening techniques, in addition to a contone dot generator. The user can select which form of outlet is desired for any situation. Additionally, the software RIP rasterizes data at a variety of bit depths, from 600x600 dpi at eight bits per pixel per color for contone output to 600x1200 dpi at one bit per pixel to be screened by a screening algorithm that is proprietary to the *Harlequin* software RIP. When rasterizing data at 1200 dpi for output at 600 dpi, the present system utilizes the extra data to modify the laster spot utilizing the print engine's ability to modulate the beam.

[0056] The software RIP is connected to the printer via a PCI interface, as described hereinabove and which will be described in more detail hereinbelow. This PCI interface will allow two engines to be connected to the PC bus and a parallel

data channel associated therewith, which provides a data transfer rate of 13 megabytes per second per engine, fast enough to handle hundreds of compressed pages a minute. The PCI bus feeding these engine interfaces operates at up to 120 MB per second. The job parser 412 is operable to split virtual print engines across multiple print stations, as described hereinabove, which in conjunction with the electronic collator 418, is operable to set up and monitor queues, support "virtual printers," and provide diagnostic feedback and status of the print stations, which is received from the print engines 408.

[0057] The electronic collator 418 is, as described hereinabove, the process of rasterizing every page in a document only once and then printing the pages in order, such that no mechanical or manual collation or sorting is required on the output of the print engines 408, this being a function of the actual way in which the print engines bins are configured. The compressed bit maps are saved, such that multiple copies of the pages can be printed subsequently at the engines' full rate of speed, this being the result of rasterizing on a page-by-page basis, with no additional rasterizing being necessary for each multiple page, *i.e.*, once a page is rasterized, it can be printed any number of times without requiring further rasterization of the original input. The job parsing operation, as described above, is operable to determine the total number of pages in the job (the number of pages in a document times the number of copies of the document in the job), and then spread the job out "equitably" among the available engines, such that the output of one engine placed on top of the other engine yields a complete job, as if it had come out of one engine. This, of course, is a function of the number of engines utilized and the way in which their output bins are disposed. However, each of the engines is combined in a particular configuration setup that will define a virtual printer that will be associated with a particular job.

[005°] Whenever a multi-page document requires multiple pages to be printed, it is first necessary to rasterize the entire document and store it in the memory 414. Thereafter, a virtual job stack is created that has one copy on top of the other, as described above with respect to the stacks 326 and 328 in FIGURE 9. Boundaries

are then defined, as noted above with respect to the "gather mode" in FIGURE 10, which defines which portion of the stack goes to which engine and the portions of the stack designated for those engines are routed to the engines via the distribution process, as described above with respect to block 14 in FIGURE 1. The key aspect of this system is that a document is rasterized only once, but emerges as multiple copies (or gathered copies) and that the job is printed at the speed of multiple engines in parallel.

[0059] The electronic collation step is completely automatic, transparent to the operator, which operator merely directs the print station as to which job to print and how many copies are needed, the print station comprising all of the printers. This creation of the job stack and defining of boundaries of the job stack for distribution to the engines is determined in a relatively short amount of time on the order of seconds. The purpose is primarily to ensure that the resources are utilized at their maximum ability. At the completion of a press run, the operator then stacks the various output piles on top of each other to complete the collated job. Of course, as will be described hereinbelow, an automatic "finishing" system could be utilized to extract the documents and put them in the appropriate stacks. It is only important that, when the stacks are defined within a given printer, there is some indication, such as a separator page, that will allow the particular stack created between separators, to be assembled with another stack from another printer in the desired print job output.

[0060] If one of the print stations, *i.e.*, one of the print engines 408, is down, the job will automatically be reconfigured such that it is parsed over the remaining print stations. An output will then be provided to instruct the operator how to arrange the pages for pickup from the output bin.

[0061] With respect to duplex printing, duplexing can be performed by a dedicated engine or, with an engine that does not have an automatic duplex feature, a "work-and-turn" procedure is utilized. In this procedure, after the first sides are printed, the lights on the print station console blink red and send a message to reload the paper.

The operator then removes the sheets printed on one side from the output of the print engine 408, turns them over and places them in the input tray. The process then prints the other side and places them into the output bins. The result is pre-sorted output of the entire job. The operator then moves from one machine to the next, putting the sets in one collated stack. If needed, slip sheets fed from the printers' manual input trays can be inserted between jobs or between copies, these being separator sheets. This technique can be facilitated, since the system has stored the individual pages and associated with those pages information regarding which job it is associated with and the page number it is associated with. Therefore, once the job is rasterized and the compressed bit image is stored, it is only necessary to extract the even-numbered pages, print them, turn over the documents, and then print the odd-numbered pages.

[0062] By providing the page-by-page basis, this allows a merge operation to be facilitated. If the user desires, the print station manager 410 can merge pages of a "form" document with pages of another job. This is to be distinguished from a conventional word processing system in that the image formed of a bit mapped image for each page can merely be printed in a predetermined order. For example, a page from a form document could be inserted between pages 10 and 11 of a single other document, such that the final document results in page 10 being followed by the form page and then being followed by page 11. This merely requires the job stack to be redefined and redefine the output job. Of course, if two jobs are input at the same time and are to be merged, it will require initial rasterizing and storing of the jobs prior to the merging operation. If the jobs have already been rasterized and pre-stored, then it is only necessary to create the virtual job stack, which is relatively quick.

[0063] The spooler 419 is operable to optimize printing over the network. The spooler can take jobs directly from a workstation and route them either to disk or directly to the RIP 416 in order to begin processing. This is primarily directed toward Macintosh computers. For general IBM-compatible PC's, the jobs will go directly through the print manager on the Windows NT print manager directly to the

spooler. After the jobs are rasterized by the RIP 416, they are then stored on the disk 414 prior to output to a print engine.

[0064]   Jobs may be printed as contone images or screened.  The spooler will publish print cues representing printer characteristics.  This will then be displayed to the user as printers in the print setup menus for Windows.  Also, different queues can be provided with other characteristics, such as different paper sizes or collating requirements.

[0065]   Referring now to FIGURE 15, there is illustrated a block diagram of the lower level architecture between the generation of a virtual printer and the print engine 408.  As described above, the virtual printer is a configuration setup in which a certain number of printers in the system are utilized together to produce a single job.  Just as queues represent characteristics that affect how the RIP will process the file, the virtual printer defines the way in which the engines are configured, which affects how the RIP will feed them with the pages to the printers.  Each "virtual printer" is actually one or more physical engines, which virtual printer essentially represents the print engines that are available to the RIP sending a document to be printed, i.e., the final destination is defined.  The virtual-printer system is a method to allow the user to define fewer than the entire series of printers for association with a single job, such that one or more engines are available to print a different job. Overall, each system and each different site may set up as many virtual printers as it desires with its allocated engines.  For example, an operator could set up two engines as one virtual printer, the other engine as a second virtual engine, and all four engines as a third virtual printer.  Then, the job could be directed from the RIP to the third virtual printer, thus using all the printers to output the same job.  Alternately, two separate jobs could be printed concurrently, each using one of the first two virtual printers.

[0066]   In the block diagram of FIGURE 15, there are illustrated three virtual printers 420, labeled virtual printer A, virtual printer B, and virtual printer C, respectively.  Associated with the virtual printers 420 are two distinct allocation

systems, an engine allocator 422 and a resource allocator 424. The engine allocator 422 is responsible for managing the physical print engine resources, while the resource allocator 424 manages the memory resources. Since disk usage, CPU usage and bus bandwidth are directly related to memory usage when sending bit maps to the engines, managing memory usage effectively manages the other resources.

[0067] In general, the engine allocation block 422 is required to give each virtual printer 420 access to the print engines 408 in a controlled and predictable manner that will guarantee maximum use of each engine. The engine allocator 422 works in concert with the electronic collator 418, while the electronic collator 418 itself generates job stacks for each engine, the engine allocator 422 ensuring that there is a physical print engine 408 available to execute a particular job stack.

[0068] The resource allocator 424 has two basic functions: memory management and performance management. If a system has only a limited amount of memory, and all engines wish to utilize this memory to load bit maps at once (which contain more data than available memory), the system may cause unpredictable behavior unless the memory is managed. In some systems, it is more desirable to dedicate more memory usage (bus CPU time and bus bandwidth) to the RIP versus the physical engine or *vice versa*. With the resource allocator 424, this is a relatively straightforward method which can be presented as a slidebar to the user. One end of the slidebar increases engine performance while the other increases RIP/spool performance. The block diagram of 15 depicts the basic communication and data flowpath from the virtual printers 420 to the print engines 408. The virtual printer A is illustrated as utilizing print engines 408 labeled PE1, PE2, and PE4, virtual printer B is designated as utilizing only print engine 408 labeled PE 3, and virtual printer C is not printing.

[0069] Each of the print engines 408 has associated therewith in the software a physical print engine object 426, labeled PPE1, PPE2, PPE3 and PPE4, respectively. The physical print engine objects 426 provide control of the engine. All access to the print engines 408 is controlled through the PPEs 426 that are associated

therewith. The PPE 426 is responsible for maintaining real-time status of the print engine and is also operable to isolate the associated print engine 408 from the remainder of the system. This permits different types of print engines to have different PPEs. New engine types can be added to the system by simply developing new PPEs. The PPE 426 maintains the physical link to the engine. A logical link is also maintained by the engine allocator 422. In this manner, if the user physically switches cables, the logical link is maintained.

[0070] Each of the PPEs 426 drives a kernel device engine 428. These are created during system utilization to develop kernel mode device objects for each printer port, with these portions then registered in the system hardware register. These kernel objects are different from PPEs. During initialization, it is a function of the engine allocator 422 to enumerate each of these physical ports, create a PPE object, and attach one per port. Once assigned a port, the PPE object then begins communication with the associated physical print engine 408 through the kernel mode device engine 428 and the associated kernel mode device object.

[0071] The engine allocator 422 will maintain a list of pointers into the PPE objects and kernel mode device objects to access status functions, as well as control functions. As such, the engine allocator 422 will be connected to the PPEs 426 through interconnections 430, which are logical interconnections to the virtual printers 420. With these connections, any routine system can request status of a physical engine 408 from the engine allocator 422. When a request for real-time status is presented to the engine allocator 422, it reads the status from the PPE 426 associated with that engine and returns to the call-in function. All status requests must go through engine allocator 422. The request can be made of logical as well as physical engines. The engine allocator 422 maintains the logical and physical mappings of each engine.

[0072] The PPE object 426 is instantiated from a derivative from the C Printer class. The C Printer class is the base class all printer objects are built upon. In a preferred embodiment, there is only a single printer class associated with the *Canon*

*P320* engine class. The interfaced printer object is the same regardless of the type of physical printer they are connected to. The PPE object isolates the rest of the system from the physical device. Each time a new device is to be connected to the system, an associated PPE class must be developed for that device.

[0073]   The kernel device engines are output to a controller object 436, which is then operable to interface a host adapter 438, there being one for each two printers 408. This is essentially the PCI adapter described above.

[0074]   The print engine C Printer class is, as described above, specific to the *Canon P320* laser engine. This class must communicate through the kernel mode driver to the physical engine. The higher levels of communication protocol are implemented in this class, while the lower level layers are implemented in the kernel device engine 428. Since the *Canon P320* does not send any unsolicited messages, the messaging protocol between the *Canon P320* class and the engine is a master/slave protocol, with the engine being the slave. However, it should be understood that solicited messages can be accommodated with a different engine. In the preferred embodiment, the *Canon P320* class must maintain a real-time engine status. This is accomplished via polling the engine every three seconds. Only changes are updated during this polling interval. Each time the *Canon P320* class detects that the engine has been powered on or reset, an entire status update is performed.

[0075]   In order to understand the engine allocation, the concept of "job stacks" will be further elaborated upon. A job stack is basically a dynamically sized array of instructions which is delivered to the PPE 426. Each entry in the array is defined by a number of variables. There is a Number of Copies variable, which instructs the PPE 426 to print each page a defined number of times before moving on to the next page. There is a Begin Page variable to define when the first page of a document is to print, and an End Page variable which defines the last page of the document for this instruction. A Command variable is also input, which is utilized for special commands, such as printing separation pages. In the following example in Table 6, a

Command value of "0" indicates nothing, while a Command value of "1" indicates that a job separator page must be printed. Job stacks are dynamic and can contain any number of entries. The example in Table 6 utilizes a ten-page document that is to be printed with four copies utilizing three engines, with a separator page between jobs feature turned on. Table 6 is as follows:

**Table 6**

|  | PPE1 | PPE2 | PPE3 |
|---|---|---|---|
| Instruction 1 | 1, 1, 10, 0 | 1, 4, 10, 0 | 1, 7, 10, 0 |
| Instruction 2 | 1, 1, 3, 0 | 1, 1, 6, 0 | 1, 1, 10, 0 |
| Instruction 3 | 1, 0, 0, 1 | 1, 0, 0, 1 | 1, 0, 0, 1 |

Examining the above Table 6, it can be seen that PPE1 will print one copy of pages 1-10, followed by one copy of pages 1-3 of a document followed by a job separator. PPE2 will print one copy of pages 4-10, followed by one copy of pages 4-6 followed by a job separator. PPE3 will print one copy of pages 7-10, followed by one copy of pages 1-10, followed by a job separator. Since the printing is face-down printing, once the job is complete, the user simply takes the pages from PPE3 and stacks them on top of the pages from PPE2 (still face down), and then takes this new stack and places it on top of the pages from PPE1 for a completed and collated job.

[0076] The engine allocator 422 provides for Controlled Dynamic Engine Allocation. The goals of engine allocation are two-fold: provide equitable (round-robin) allocation of engines, while at the same time maximizing the use of individual engines. Controlled engine allocation means that virtual printers consist of predefined sets of print engines and cannot utilize engines outside of the set with the predetermined job stack for a physical engine unable to be changed "on-the-fly."

[0077] As an example, if Virtual Printer A consists of physical engines PE1 and PE2, it can never utilize physical engines PE3 or PE4, unless the system is reconfigured. As an example, if a virtual printer consisted of three print engines and

was presented with a one-page job, it would not be efficient to tie up all three physical engines to complete this one-page job. In this case, the virtual printer 420 would submit the job to the electronic collator. The electronic collator would then return only one job stack. Since there is only one job stack, the virtual printer 420 would request only a single one of its engines. A more complicated case to this is one that will be more common. Consider the case of two virtual printers, Virtual Printer A and Virtual Printer B, and four print engines, PE1, PE2, PE3, and PE4. Virtual Printer A consists of physical engines PE1 and PE2, while Virtual Printer B consists of physical engines PE2 and PE3. If a 100-page job is assumed, which 100-page job has been split up by the electronic collator into job stacks of 50 pages each, it is then necessary to process these with the virtual printers 420. If the jobs arrive at each of the virtual printers at the same time, one job for one of the virtual printers 420 and one job for another of the virtual printers 420, since only one virtual printer 420 can utilize PE2, the other must wait. In this case, assume that Virtual Printer A received the job first, such that PE1 and PE2 are printing the job for Virtual Printer A. If there were no dynamic allocations provided by the engine allocator 422, PE3 would remain idle until PE2 was released. This is not desirable, since there may be more jobs behind the two mentioned, and PE3 is sitting idle, *i.e.*, system resources are not being utilized at their maximum potential. If a user walked up to the console, he would then see several jobs waiting and an engine sitting idle.

[0078] In the controlled dynamic allocation provided by the engine allocator 422, the following sequence of events would occur. First, engines PE1 and PE2 would begin printing their 50-page job stacks for Virtual Printer A. Engine PE3 would begin printing its 50-page job stacks for Virtual Printer B for the portion that has been designated to be printed by PE3. At a later time, PE1, PE2, and PE3 would complete their job stacks at about the same time. Once PE2 is released from Virtual Printer A, it can then begin printing its job stack that was designated for being printed by Virtual Printer B. At this point, engines PE1 and PE3 are available for other jobs, thus maximizing throughput. An important rule to note is that PE2 will print its jobs for Virtual Printer B in their entirety. Although it may seem more efficient to send a portion of the PE2 job stack to PE3 once it is released, this may

not be the most efficient method for a given mode, since this would mean splitting the PE2 job stack real-time and printing part of it on PE2 and part of it on PE3. This does not provide significant advantages and can cause some collation problems when a user stacks the documents after the job has been completed. If additional jobs are waiting, the engine usage is the same and no benefit is gained. If this approach were taken, the stacking order would be disrupted in an unpredictable manner for the user, since the PE2 job stack is partially printed on PE3. However, with the use of page separators and indicators, this could be achieved. Further, if a more sophisticated finishing system other than a user stacking documents were utilized, this could be facilitated.

[0079] Each virtual printer is allowed to only have one request into the engine allocator 422 at a time. If the engine allocator 422 receives a request from a virtual printer that has already queued a request, then the request fails immediately. The requesting virtual printer must pass several bits of information to the engine allocator. For each physical engine the virtual printer wishes to utilize, it must supply a handle for the physical printer (this was obtained by performing logical to physical mapping), and a handle of an event object to signal when the engine is available. This is information that is passed in as a pointer to a list of request structures in the engine allocator 422.

[0080] It is the task of the virtual printer 420 to create a "thread" to pass the job stack to the PPE 426, these threads illustrated in FIGURE 15. This thread will wait on an event that is signaled by the engine allocator. The engine allocator 422 will take the request and sort it into a separate queue for each physical engine. As soon as an engine becomes available, the engine takes the next request out of the queue and signals the event object via a SetEvent. If the SetEvent function fails and the engine allocator 424 assumes the virtual printer 420 no longer requests the engine, the engine allocator 422 will move on to the next request in that queue. There is a separate queue for each physical printer PE1-PE4, and the queues are independent of each other. Each queue entry will consist of a handle for the virtual printer ID and the handle for the event handle.

[0081] Once the virtual printer has completed a job stack on a particular printer, it must then free the physical printer by calling an engine-free handle. At this point, the engine allocator 422 will move on to the next request in that engine's queue.

[0082] Engines in an Error state are treated no differently by the engine allocator 422. It is the task of the virtual printer 420 to decide what to do with an engine in an Error state. Virtual printers can still request and be granted use of the physical engines in the Error state. If the virtual printer has no use of an engine in an Error state, it can then release this physical engine back to the system immediately. If the virtual printer determines that the Error condition will be corrected, then it can hang on to the engine and release it later. A virtual printer does not have to request all of its physical engines to print a job. If a virtual printer 420 is configured with three physical engines and only receives two job stacks back from the electronic collator, then it has the ability to only request two engines from the engine allocator.

[0083] Referring now to FIGURE 16, there is illustrated a flowchart depicting the engine allocation operation for setting up the requests in the queue. This is initiated at a block 440 and then proceeds to a block 442 to determine if a request has been received from a virtual printer. If not, the program will flow back to the input thereof from the "N" path. If a request has been received, the program will flow to a decision block 444 to determine if there is an existing request for that virtual printer in the queue. If so, the program will flow along a "Y" path to a block 446 to fail the request and then back to the input of decision block 442 to wait for another request from a virtual printer. If this is the only request by the virtual printer and it is not currently serving a request from a virtual printer, the program will flow from decision block 444 along the "N" path to a function block 440 to receive the handle for each print engine defined as being associated with the virtual printer. The program will then flow to the function block 450 to receive the event object and then to a function block 452 to queue the request for each print engine. The program will then flow back to the input of decision block 442.

[0084] Referring now to FIGURE 17, there is illustrated a flowchart depicting a portion of the engine allocation operation associated with servicing the request in the queue. This is initiated at a block 454 and then proceeds to a function block 456 to service the request in the queue for a given one of the engines. The program will then flow to a decision block 458 if the SetEvent has failed. If the SetEvent function fails, then the engine allocator will assume that the virtual printer no longer requests the engine and will flow along the "Y" path. If it has not failed, this indicates that the virtual printer still has possession of the associated physical print engine and then flows along a "N" path to decision block 460 to receive the FreeEngine handle from the virtual printer. This handle is generated once the virtual printer has completed a job stack on a particular printer. If not, the program will flow along the "N" path back to the input of decision block 450. If the engine is determined to be free, the program will flow along the "Y" path to a function block 462 to service the next request. The "Y" path from decision block 458 will also flow to the input of function block 462. After servicing the request, the program will flow back to the input of function block 456.

[0085] Referring now to FIGURE 18, there is illustrated a flowchart depicting a portion of the engine allocation operation associated with the status request. This is initiated at a function block 464 and then flows to a decision block 466 to determine if a request for status information on a given print engine has been received. If not, the program will flow back to the input of decision block 466. If so, the program will flow along a "Y" path to a function block 468 to retrieve the status of the print engine and then to a function block 470 to forward the status to the calling routine. The program will then flow back to the input of decision block 466.

[0086] Referring now to FIGURE 19, there is illustrated a flowchart for the resource allocator. The program is initiated at a block 472 and then flows to a decision block 474 to determine if a request has been received from the PPE object. If so, the program will flow along a "Y" path to a decision block 476 to determine if there is available memory. If not, the program will flow along an "N" path to a function block 478 to instruct the engine allocator 422 to queue requests until

memory is freed. If memory is available, the program will flow from the decision block 476 to a function block 480 to decrement the available memory and then back to the input of decision block 474.

[0087] If decision block 474 had determined that no request was received from the PPE, the program will flow along the "N" path to a decision block 482 in order to determine if an indication has been made that memory was freed from one of the PPEs. If not, the program will flow along the "N" path back to the input decision block 474. However, if additional memory had been freed from one of the PPEs, the program will flow to a function block 484 to increment the memory and then to a function block 486 to send an instruction to the queue. Basically, when a job stack is forwarded for printing and memory is unavailable, the resource allocator will place that job in a queue and it will remain in the queue until the memory is available. When the memory is available, the queue is served on a first-in-first-out basis. The program will then flow back to the input of decision block 474.

[0088] Referring now to FIGURE 20, there is illustrated a block diagram of a prior art interface for a conventional PCI bus. CPU 490 is typically provided at the heart of any computing system. This typically will interface through a system bus 492 to a PCI device 494, which then interfaces to a PCI bus 496. The system bus 492 is typically a 64-bit bus, whereas the PCI bus 496 is a 32-bit bus. The PCI bus operates with a PCI protocol at a rate of 132 MegaByte/s. The PCI bus 496 is then interfaced with a PCI interface device 498 to an I/O bus 500. This then interfaces through an I/O chip 502 in a conventional manner to a parallel cable 504 and then to a printer 506. Thereafter, the printer operates in a normal convention. This normal convention is that the parallel cable is interfaced through an I/O device 508 to output data to an internal printer RIP 510 which rasterizes the data and outputs it to a memory 512 for subsequent input to the marking engine 514. This, again, is conventional in the printer 506. The PCI device 494 is basically configured of a PCI chip set which is conventional for converting the data and control information on the system bus to information in the PCI protocol. The PCI interface 498 is basically a bus mastering/bus interface. In the present invention, this utilizes a PLX9060,

manufactured by *PLX Technology*. In effect, PCI interface 498 and the overall PCI

protocol allows data to be accepted in large bursts, thus providing relatively high

throughput for data as compared to a general parallel I/O system. However, in the

prior art system of FIGURE 20, the RIP 510 is disposed in each printer 506, such

that a PCI interface 498 is required for each printer and the data must be transferred

to the printer in the form of a multi-page document, RIPPED in the printer and then

output to the marking engine 514.

[0089]    Referring now to FIGURE 21, there is illustrated a simplified block

diagram of the interface of the present invention. A PCI interface device 516,

similar to the PCI interface 498 in that it utilizes the same chip, is provided for

interfacing with a PCI bus 518. The PCI interface then is shared by two host

adapters 518 for two different printers 520. The host adapter 518 interfaces with a

proprietary cable 522 and then to a print adapter 524, which is then interfaced to the

printer 520. The print adapter 524 basically controls transfer of data from the cable

522 to the printer and controls the operations of the printer through various

control/status lines. Status information can be returned from the printer to the PCI

bus 518.

[0090]    Referring now to FIGURE 22, there is illustrated a block diagram of the

host adapter. The PCI interface 516 is operable to interface data from the PCI bus

518 to an intermediate bus 526 which is a 32-bit bus. The data is then input to a

FIFO 528, which FIFO 528 is operable to receive the data at a 32-bit word length

and output the data at an 8-bit word length on an 8-bit bus 530. This is essentially a

double word (32-bit) to byte (8-bit) funneling FIFO. The order of the double word to

byte funneling is least significant, D7:0, first, D15:8 next, D23:16 next, and D31:24

last. The D31:0 bits have a 1-to-1 correspondence to AD31:0 of the PCI bus.

Implicit in this process is that all image data (even if it is compressed image data)

must be transferred to the host adapter in multiples of double words. The 8-bit bus

530 is then input to a set of drivers 532 which will then drive the cable interface 534

to output data on a data bus 536, which forms a portion of the cable 522.

Additionally, control status information is transferred over a plurality of the

control/status lines 538 wherein directional drivers 540 are provided for interfacing with a UART 542. The UART is then operable to interface with a status control block 544 to interface with the bus 526 to allow communication of the status control information between the PCI and the control/status lines 538. A timer 546 is provided for controlling the timing functions for the system. The differential drivers and receivers utilized in the host adapter are the type DS903C031 and DS903C032, respectively. These devices are manufactured by *National Semiconductor*. The cable interface 534 is a 50 position AMP 0.8 mm CHAMP, Part No. 78796-1, with a cable 522 being a custom 16 individually shielded twister pair cable of eight meter length.

[0091]    Referring now to FIGURE 24, there is illustrated a block diagram for the print adapter 524. The cable 522 is interfaced through a driver/receiver 550 to a FIFO 552. The FIFO 552 is operable to provide an elastic storage capability which is then input to an internal data bus 554. The internal data bus 554 then interfaces with an unpacker/unloader 556, which is operable to retrieve the data from the FIFO 552 and then decompress this data. The entire operation is controlled by a CPU 560, which CPU 560 is operable to control the number of bits per pixel in the unpacking operation. This is then input to a transform block 558, which transform block 558 is operable to perform a calibration adjustment. As will be described hereinbelow, the engines in a given virtual printer are "color balanced." In order to do this, each engine is calibrated and compared to an internal master color space. The data that is transferred to the FIFO 552 is formatted in this master color space. Any aberrations of the printer due to parameters associated with a given engine that may yield to wear, etc., can be compensated for in this calibration procedure. Once calibration is complete, a look-up table 562 is loaded with calibration information, which calibration information is then utilized by the transform block 558 to correct the color space. This data is then input to the marking engine 514.

[0092]    The CPU 560 also interfaces with the marking engine 514 through a control/status bus 564. This control/status information can then be read by the CPU 560 to a UART 566, which interfaces with the cable 522 through the driver/receiver

550. Control information can then be transferred between the cable 522 and the CPU 560, such that the marking engine 514 can be controlled and status information requested from the marking engine 514.

[0093] Each print adapter connects to a host adapter using a custom cable assembly that consists of 16 individually shielded balanced twisted pair conductors (one unused pair).

| Name | No. Of Pairs | Input/Output (@HostAdapter) |
|------|--------------|------------------------------|
| PxD7:0 | 8 | Out:Date |
| PCLKx | 1 | Out:Clock |
| PWEx | 1 | Out:Printer Write Enable |
| HTXDx | 1 | Out:Host Tx (status) |
| HRXDx | 1 | In:Host Rx (status) |
| PAFx | 1 | In:Printer Almost Full Flag |
| EOPx | 1 | In:End of Plane (Data extracted out for page -- used for page sync) |
| FLUSHx | 1 | Out:Resets state of print adapter |
| | 15 | |

x=>A or B

HostAdapter to PrintAdapter Cable Differential Pair Definitions

PxD7:0+

PxD7:0-     Printer Image Data, D7 is MSB, D0 is LSB, sense is positive true, and is clocked into the PrintAdapter buffer FIFO on the positive transition PCKx.

PCKx+

PCKx-         Printer Buffer FIFO Load Clock, used to clock image data into
              PrintAdapter Buffer FIFO.  Setup and hold times of image
              data are reference to the positive transition of this signal.


PAFx+

PAFx-         Printer Buffer FIFO Almost Full Flag, sense is negative true.
              The HostAdapter will stop transmitting data and de-assert
              PWEx within 4 positive transitions of PCKx.


PWEx+

PWEx-         Printer Buffer FIFO Write Enable, sense is negative true.  This
              signal is used to enable the loading of a synchronous FIFO on
              the PrintAdapter with image data.  When this signal is
              asserted, image data should be accepted by the PrintAdapter
              on the positive transition of PCKx.


HTXDx+

HTXDx-        HostAdapter Asynchronous Start/Stop Transmit Data, sense is
              positive true, i.e., the start bit is a logical flow.  The
              asynchronous data rate is 9600 Baud.  The form is one stop bit
              and no parity.


HRXDx+

HRXDx-        HostAdapter Asynchronous Start/Stop Receive Data, sense is
              positive true, i.e., the start bit is a logical low.  The
              asynchronous data rate is 9600 Baud.  The form is one stop bit
              and no parity.


EOPx+

EOPx-         End of Plane Pulse, sense is positive true.  Asserting this
              signal for at least 3 positive transitions of PCKx will result in
              an interrupt being generated on the PCI bus by the

HostAdapter (interrupt support logic must have the EOP interrupt unmasked).

FLUSH+

FLUSH-     Printer Buffer FIFO and Support Circuitry Reset, sense is negative true. This signal should be used by the PrintAdapter to flush the image data FIFO and to re-initialize image generation circuitry.

[0094]   Referring now to FIGURE 24, there is illustrated a timing diagram illustrating a typical host adapter unload sequence. The frequency of the clock is 13.75 Megahertz.

[0095]   As will be described hereinbelow, there is a command FLUSHx, which is an output from the system to the printer that is operable to reset the state of the print adapter. This is utilized for a situation where there is no page synchronization. The FLUSH operation is operable to flush the FIFO 552 in the print adapter between pages, such that there is a clean state. This occurs at the end of a plane (or page) after the EOPx signal is generated by the print adapter. With page synchronization, on the other hand, information is provided that indicates that the FIFO 552 is flushed and the print adapter is ready to receive a new page of information. Page synchronization also provides for counting the number of pages to determine how many pages have been printed. If a page is not received in a predetermined amount of time, then an Error condition is generated. At the EOPx, the system then looks at the FIFO 552 to determine if it is empty. If it is not empty, this indicates that all the information has not been transferred from the FIFO and, thus, an error is indicated. In any event, this indicates to the system that a given page has not been printed and either this page needs to be reprinted or sent to another engine. In order to facilitate this page synchronization, of course, it is necessary to know what the status of the printed page is when the end-of-page signal is generated. If page synchronization is not utilized, the FLUSH signal is utilized to ensure that an error occurring on one page does not carry over to the next page.

[0096] Referring now to FIGURE 25, there is illustrated a block diagram of a prior art method for handling color spaces. CPU 580 is provided which interfaces with a number of different color devices, a scanner 582, a display 584, a film recorder 586 and a printer 588. Due to color balancing, most systems have used what are referred to as "device profiles" which resulted from a color management system specification that was created by the International Color Consortium (ICC). These device profiles are typically generated by the manufacturer, or in some cases, by the user, which describes the color characters of a particular device. Therefore, each of the devices 582-588 would have a device profile associated therewith, which is stored in a block 590. These device profiles are provided for allowing an input from a scanner 582 to either be displayed on display 584 or to be printed on printer 588, with a translation being performed therebetween that utilizes the color independent space as the intermediate. The actual transformation is performed by the CPU 580, which reads the device profile link and performs the mathematics necessary to transform from one native space to another. This, of course, occurs on the user's machine and it results in a significant speed impediment.

[0097] Referring now to FIGURE 26, there is illustrated a block diagram of the prior art process flow. The input device that is being mapped to the output device is illustrated as having a first color space in a block 592. This is then mapped through the device one color profile in a block 594 to a master color space, the XYZ color space, in a block 596. This is then mapped through the device two color profile in a block 598 to the device two color space, in a block 600. At this point, the color space should be acceptable for printing and it is then routed to the printer through the RIP 602 and to the marking engine 604. However, it is noted that this must be performed prior to the RIP operation, which RIP will rasterize each of the pages for input to the marking engine 604. However, there is one significant disadvantage to this type of operation when processing pages in accordance with the multiple printing virtual print engine concept described hereinabove. This is the fact that, first, the operation is performed prior to rasterization and, second, that it must be performed for each page; that is, each time a job is created and sent to the printer, the

translation must be performed. Further, when utilizing multiple printers, as in the present system, then this matching or balancing must be performed for each printer. This creates some difficulties, in that the printers are not necessarily defined prior to the RIP operation but, rather, after the pages are ripped and stored.

[0098] Referring now to FIGURE 27, there is illustrated a block diagram of the balancing method of the present invention. The input device exists in the device one color space, as represented by a block 606. This is mapped to a system color space in a block 607 through a color matching algorithm in a block 608. This matching algorithm is basically that described above with respect to the operations performed between blocks 592 and 600 in FIGURE 26; that is, device one color space is first converted to the master color space, the XYZ color space, and then mapped from the XYZ color space to the system color space in block 607. The system color space, in the present invention, is basically the device profile of the generic printer that is utilized in the system. Therefore, if the Canon C320 engine were utilized in the system, then that would constitute the system color space. Therefore, the device one color space would be mapped through its device one color space to the master color space and then through the Canon C320 color profile to the system color space. This constitutes a reference color space. Of course, if all of the engines are calibrated and operated identical, then any engine in any Canon C320 engine on the system would print identical to another. However, this is not the case, since all engines print slightly different. As such, each would have a slightly different color profile which would have to either be iteratively determined or defined by the manufacturer. This is difficult if not impossible to implement.

[0099] In the present system, the system color space is defined as being that for a standard printer, which is defined prior to the RIP operation. During the RIP operation, in the preferred embodiment, the color matching operation performed is a part of the RIP operation, such that after going through a RIP operation, as illustrated by a block 610, there will be rasterized images for each page of a document that have been color balanced to the system color space. This rasterized image will then be routed to the desired engine via the methods described hereinabove with respect to

the virtual printers and physical print engine objects, etc. The marking engines in FIGURE 27 are illustrated as being three marking engines 612, 614 and 616. Each of the engines 612-616 has associated therewith a color mapping block 618, 620 and 622, respectively. Each of these color mapping blocks 618-622 is operable to adjust the color of the bit mapped image that is forwarded thereto to account for aberrations in the marking engines 612-616 as compared to the standard engine which was utilized to generate the device profile used to convert all color spaces to the system color space. This color mapping function is a calibrated function, which will be described hereinbelow. This is facilitated with the use of the lookup table 562 described herein above with reference to FIGURE 23.

[0100] The color mapping devices 618-622 allow each of the engines 612-616 to be mapped to the system color space after the RIP operation. The mapping process, as will be described in more detail hereinbelow, first renders the engines linear in response, such that similar changes in density are achieved for similar changes in data. This is a combination of an internal linearization algorithm that runs on power up and an additional linearization added. This process utilizes a manual gamma calibration which is facilitated by setting the lookup table to a 1:1 ratio with no calibration difference between the system color space and that of the printer and then running a test pattern therethrough. The test pattern is then examined with a colorimeter to determine if the printed image represents the image that should have been printed. If not, an offset is determined and stored in the lookup table. For contone images, there are typically 256 values. Each of these values can have a mapping value associated therewith, such that when a bit value is input to one of the color mapping devices 618-622, the value in the lookup table is an output. For example, if a value of 255 were input and the lookup table determined that it should be 217, then a value of 217 would be output to the marking engine.

[0101] Referring now to FIGURE 28, there is illustrated a flowchart depicting the operation wherein the lookup table is created. The flowchart is initiated at a block 626 and then proceeds to a block 628 to set the lookup table to a 1:1 ratio, such that when an 8 bit value is input to the transform block 558 of FIGURE 23, the same 8

bit value is output. The program then flows to a function block 630, wherein a test pattern is set to the output device and printed. This test pattern consists of four bands of 256 steps each, each step being a different density of toner, associated with each of the expected 8 bit values. These 8 bit values extend from 0 to 256. In general, the 8 bit values for each of the colors cyan, magenta, yellow are equal. The program then flows to a function block 632, wherein this pattern is read into the colorimeter. The colorimeter is of the type X-rite DTP51 colorimeter. This is to be distinguished from a densitometer, which measures only the reflectants from a surface to determine density. The colorimeter is based upon measurement of wavelength and determines the percentage of the XYZ space, this being "device independent" color. As such, different devices can have the same density reading with a densitometer, but a different percentage reading with the colorimeter.

[0102]    The colorimeter will basically determine a value for each of the spaces and provide an output therefor. The program will then flow to a function block 634 to compare the output of the colorimeter to the system color space values that would be expected to be on the colorimeter. There is a standard table that is provided for the colormetric values that are associated with an 8 bit input value. For example, if the maximum density at the 8 bit value of 255 were found associated with that generated when an 8 bit value of 217 was input to the printer, then one would expect that the generation of an 8 bit value of 217 would result in the correct output color. This would in effect be the mapping function which is determined by taking the difference in a function block 636 and then creating the mapping values in a function block 638. These mapped values are then downloaded to the print adapter in the lookup table 562, as indicated by a function block 640, and then the program flows to an end block 642. For each printer, this procedure can be followed to determine the lookup table values, which lookup table values will operate on each "dot" that is to be generated by the printer and at each color. The table that is generated for the colormetric values is already linearized in the generation of that table, which table was generated in an iterative manner. The lookup table 562 in conjunction with the transform 558 can therefore provide a constraint on the output density as a function of the bit value input.

[0103]    In certain situations, although the above calibration operation has been performed, this calibration does not accurately provide for "true" gray levels. In order to provide the true gray levels, an additional adjustment is required. This adjustment, unlike the above correction for the RIPing operation, is actually a hardware adjustment on each of the engines. Each of the engines has provided thereon an adjustment for the individual toner density level. In the preferred embodiment, a Canon 320 engine is utilized to provide this capability, wherein this allows for plus and minus four ranges of adjustment from a central zero value. In order to determine how this should be adjusted, an empirical technique is utilized wherein a test pattern is run with varying adjustments associated therewith and then the most visually perceived gray setting is chosen and then made to the machine.

[0104]    Referring now to FIGURE 28a, there is illustrated a portion of a test pattern that is utilized to provide the above noted adjustment. The test pattern is comprised of a plurality of groups of images, each group consisting of three images. The three images in each group consist of a 75% contone image, a 50% contone image and a 25% contone image. There is provided one group in the center thereof that has the values selected such that they represent what an engine would appear if the density level were set to the "0" value for the rear settings on the panel. Of course, the RIP engine deals in values from 0-255 and does not have any information as to what the rear panel value correlates to. Therefore, a mapping procedure must be resorted to. Essentially, a pattern is devised that is run through the printer with all of values for cyan, magenta and yellow on the back panel set to the central "0" value. This pattern will have select values from 0-256 for the 75% contone image, 50% contone image and the 25% contone image and various combinations thereof to provide "offsets." Thereafter, the engine setting is incremented by one or decremented by one and then the pattern run again. The two patterns are compared to each other, such that one can then tell what the bit offset value is between the two patterns for the increment. This can be done for all possible settings combinations available from the back panel of the print engine 651 and it should be understood that any accessible adjustment is contemplated in this operation. This is essentially a

mapping function for each percent value. A curve of this is illustrated in FIGURE 28b which illustrates the bit value offset for the various increments. Once this mapping function is determined, then it is possible to create a 75% contone image, a 50% contone image and a 25% contone image with any combination of the increments. In the illustration of FIGURE 28a, there is one group that has no variation and one group with only a variation of -2 cyan with the magenta and yellow remaining at the central position the -2 representing two negative increments on the rear of the machine. Another group is set with a +2 setting for the cyan, with the magenta and yellow remaining the same. This can be continued for as many patches, including the center patch, as is deemed necessary to represent all relevant variants from the center patch.

[0105]    Referring now to FIGURE 28c, there is illustrated a flowchart depicting the above noted adjustment operation. The program is initiated at a start block 641 and then proceeds to a function block 643 to run the contone pattern on all engines. This will provide a sheet with the pattern in accordance with FIGURE 28a. The program will then flow to a function block 645 to select the ones of the patterns that have the most appropriate gray level. To facilitate this, one engine's pattern is viewed and the observer then selects the pattern that he or she perceives to be the best gray pattern. Once this is selected, this pattern is cut out and the value of that pattern then applied to that particular engine. This cut out pattern has holes punched in the center of each of the three patches, the 75% patch, the 50% patch and the 25% patch, and then this patch overlaid on the groups of three patches on the other test patterns of the other engines. This is for the purpose of ensuring that the perceived gray level on one is the same as the selected gray level on the others. Once the gray level on the other engines is selected, these settings are input to the back of the engine for the density level controls for each of the cyan, magenta and yellow density levels.

[0106]    The above noted color balancing calibration procedure is acceptable for color balancing contone images, but there are some aspects of halftone rendering, for example, bi-level and quad-level techniques, that may provide a problem, since the different levels are achieved by spacing dots over a given area, each of the dots

generated at the maximum pixel value. Therefore, there is an additional adjustment that is provided for bi-level and quad-level printing formats. This is an adjustment to the maximum pixel value of 255 that is generated, this defines the maximum density value for the printer. It is noted that the maximum density value may already have been adjusted for and stored in the color map block 618-622 associated with the marking etch 612-616. This is the calibration operation already performed to this point. However, there still exists a problem in that the calibration operation at the maximum density is less than adequate. The reason for this is due to the fact that at maximum density the colorimeters utilized to measure high color density level may suffer errors due to gloss variations. Further, visual inspection at high color density levels is unreliable whereas measurement at lower density levels is subject to extrapolation errors. However, it is noted that visual inspection at low color density levels is reliable since the eye can determine color shifts at these low density levels. However, the question is whether the visual inspection at low density levels can be extrapolated to a color correction at a high density level.

[0107] In order to determine how this density value is to be adjusted, a test pattern as illustrated in FIGURE 29 is developed. It essentially is a series of patches with the center patches at a location 644 being at a 50% level such that they are equal portions of cyan, magenta and yellow. These, in an ideal printer, would provide a gray patch. However, if there are any aberrations in the printer, this gray will not be true gray. The patches are adjusted, such that as the patterns move to the right in the center location, a patch 646 will be present with 50% cyan, 51% magenta and 50% yellow. The next patch to the right, a patch 648, will have 50% cyan, 52% magenta and 50% yellow. If one moves to the left in the center row, the first patch, a patch 650, will have 51% cyan, 50% magenta and 51% yellow. The next patch in the center row, a patch 652, will have 52% cyan, 50% magenta and 52% yellow. This gradual change will be noticeable to the eye and the user need only select which patch appears to be the true gray. This is then input to the system and the system makes an adjustment in a percentage of the density, which will be provided as a compensation factor in the lookup table whenever halftoning is utilized.

[0108]   Referring now to FIGURE 29a, there is illustrated an alternate embodiment for determining the adjustment to be applied to the density value. There are provided sets of patches, each comprising three patches, a 75% patch, a 50% patch and a 25% patch. Each of these patches is halftone patch at either a bi-level or a quad-level. There are illustrated only three sets of patches, whereas in the actual complete pattern there would be sufficient upwards patches to provide a complete representation of the possible color shift combinations, of which there would be a center patch wherein for each dot the density level of cyan, magenta and yellow would be the same. This is represented by the center three patches in FIGURE 29a which have a reference of "0" as a value, indicating equal amounts of the toner. The other patches are adjusted such that they will be adjusted upwards by ten point increments, i.e., they will be adjusted up by ten points in any combination. In the illustrated one, the magenta for all three patches in the left grouping is adjusted by a value of +10 and the value of yellow is adjusted by a value of +10. It is noted that, even though these are maximum at the onset, the color correction will actually provide a lower value. If the value is already at a maximum value of 255, then the value may be adjusted down relatively by a value of "-10." It is noted in parenthesis that if this were the desired gray level, then the offset to be added to achieve this patch would be a value of -10 for cyan, "0" for magenta and "0" for yellow. By varying in increments of 10, various combinations can be provided.

[0109]   In order to make the decision, the technique is for a user to run the test pattern on all the engines and pick an engine that has what appears to the viewer to be a gray level with no color shift associated therewith. This gray level to the user is now a reference. The user then cuts these three patches out and punches a hole in the center of the three patches. This patch and its associated values are then used for that engine to provide the adjustment therefor and then this patch is used for the other test patterns on the other engines to determine which group of three patches will most closely match this reference. When this is visually matched, then that particular offset value is used for that particular engine. The reason for utilizing the patch from one engine as a reference to the other engine is that there may be some perceived visual relationship between gray levels on a given sheet of paper that in

and of themselves will create an error between multiple engines. This procedure removes that source of error and essentially provides the same gray reference for all engines.

[0110] Referring now to FIGURE 30, there is illustrated a flowchart depicting the above noted operation with respect to the calibration procedure for bi-level and quad-level. This program is initiated at a block 654 and then proceeds to a block 656 to create the pattern for the bi-level test. The program then flows to a function block 658 to visually examine the patches for the most perceptively gray patch. The program then flows to a function block 660 wherein the correction factor is determined for each color at maximum density. This correction is then stored as an offset for maximum density at the bi-level and quad-level formats, as indicated by a function block 662. The program then flows to an end block 664.

[0111] When dealing with halftoning in a multiple print engine, some corrections must be provided for the variation in the "dot gain" of the various engines. Halftoning is a well known technique that determines gray levels by spreading maximum density dots out over different areas with different levels of white disposed therebetween. An example would be an 8-bit per pixel image (256 possible gray levels per pixel) rendered as a 1-bit per pixel image (2 possible gray levels per pixel). Gray is simulated on the output by patterns of dots (screens) of varying size, position, or density. To the viewer, the screens are small enough to simulate the visual appearance of the gray desired. A simple example is a checkerboard pattern of white and black squares. If the board is moved far enough from an observer, the observer sees only a middle gray (50%) tint instead of white and black squares.

[0112] The creation of screens from continuous tone data to provide a halftone image is the responsibility of the RIP. However, it is not uncommon in the industry for a 50% screen generated by the RIP (the checkerboard pattern) to be output from any given print engine as a 30% to 40% screen. This is due to various imaging non-linearities in the print engine or any type of output device such as an image setter. FIGURE 31 illustrates the kind of response typically seen from a non-linearized

output device such as a printer. The percent gray requested is illustrated on the x-axis and the percent measured is illustrated on the y-axis. It can be seen that a straight line will represent the desired with the actual curve being different. Further, each output device can have a distinctly different output response. For a given single output device, the method of correction would be to provide a correction curve that essentially mirrors the actual curve, such that an offset is provided for each point on the curve. These are referred to as "dot linearization curves." It is noted that these curves are specific to each output device, such that a higher percent gray than requested will be produced by the RIP which, when output, will provide the desired percent gray.

[0113] Referring now to FIGURE 32, there is illustrated a diagrammatic block diagram of the operation for correcting for dot gain in various output devices. Although, as noted above, a dot linearization curve can be generated for a given RIP and a given output device, a problem exists when dealing with multiple printers. This is due to the fact that each printer or output device will have its own dot linearization curve and it can be distinct. Therefore, by creating a single dot linearization curve for single engine and translating this over multiple engines, there is a strong possibility that there will be a "gray shift" between engines. In FIGURE 32, there is illustrated a single RIP 649, which is described hereinabove as a software RIP. This RIP will create a stored image which is stored in a memory (not shown). For the purposes of this description, the entire RIP operation and storage operation will be represented by the block 649. There are also provided four print engines (PEs) 651, each being a different engine. The RIP 649 in a halftone operation will be passed through a correction block 651 to provide a correction therefor, such that when a 50% RIP is requested, the overall RIP operation constituting the blocks 649 and 651 will result in a different output RIP. Although illustrated as two blocks 649 and 651, in actuality the RIP merely changes the percentage that it RIPS. Once RIPed, this is output to a job distributor 653, which is described as the job manager herein, and then distributed to the various print engines 651. The outputs of the print engine 651 will be measured, typically through a visual measurement, with the use of a colorimeter 655. The colorimeter 655 will determine for each color the percent

array for a given patch. This will be input to a linearizer 657 to provide dot linearized curves for each engine for each color, cyan, magenta and yellow. This is then input to an averaging block 659 to average all linearization curves for each color and then input this to the correction block 651. The correction block 651 in general, provides the necessary offset. As an example, consider that there are four print engines with the 50% level resulting in an actual output of 40%, 42%, 44% and 46%. The average of these four numbers would be 43%. Therefore, the correction block 651 would request the RIP 659 to provide a halftone image of 57% rather than 50%. This would result in a perceived average 50% gray image from the print engines 651. This is to be distinguished from the operation above wherein each print engine had the contone calibration disposed therein. That, of course, was directed toward density, whereas the present system is directed toward the problems with "dot coverage" in halftone job. As noted above, each system will have different parameters which are the result of various inherent properties of the particular print engine and to how it generates a particular dot and whether the dot is generated at the appropriate size to provide the appropriate cover. The RIP 649, of course, assumes that every engine operates as an ideal engine. Since it does not, it is necessary to provide this correction at the RIP level, rather than at the print engine level.

[0114]    In certain situations, there are output devices that have dot linearization curves that are outside of the range. If this engine were utilized in the averaging operation, it would essentially pull all the other engines off to some extent. In this case, an aberrant engine would be determined in the averaging operation and then a different operation provided thereby. It has been noticed through impirical tests that the print engine will have a varying dot gain or linearization curve as a function of the density control on the engine. In the preferred embodiment, a Canon 550 engine is utilized which has a software density control. This is facilitated by inputting to the engine a software value which can be stored in the engine, this being a capability of the engine. By varying this software density control, the linearization curve can be varied. Rather than utilize the linearization curve that is output from the aberrant engine, the aberrant engine is excluded from the averaging operation, an average

determined and an offset therefrom determined. This offset is utilized to drive the aberrant engine with the density control then utilized to vary the linearization curve thereof to bring it within the bounds of the non-aberrant engine. This is facilitated by density control 661 which receives from the averaging information the information as to which engine is the aberrant engine and then when the correction is applied in the block 651, the density control value is applied to that engine. This can be done on a job basis and even on a page basis when there is a mixture of contone and halftone.

[0115]   Referring now to FIGURE 33, there is illustrated a flowchart depicting the operation of performing the linearization curve. The program is initiated at a start block 663 and then proceeds to a function block 665 to run a test pattern consisting of patches of known screen percentages. Each test pattern is then read with a colorimeter or some similar device, as indicated by a function block 666 to determine the percent of gray that is actually output. The program then flows to a decision block 667 to determine if there are any aberrations. This is basically an operation wherein the value is compared with a threshold. If there are aberrations, the program will flow to a function block 668 to exclude any aberrant engine from the averaging operation and, if there are no aberrations, the program will flow along the "N" path to the output of function block 668, both block 668 and the "N" path flowing to the input of function block 669 to provide an averaging operation of all non-aberrant engine dot linearization curves. The program then flows to a function block 671 to create an average linearization curve for the non-aberrant engine and then to an End block 673.

[0116]   Referring now to FIGURE 34, there is illustrated a flow chart depicting the actual RIP operation. The program is initiated at a start block 673  and then proceeds to a decision block 675 to determine if the operation is a halftone operation. If not, the program will flow along a "N" path back to the input of decision block 675. In the halftone operation, the program will flow along a "Y" path to a function block 677 to modify the RIP operation with the averaged linearization curve as an

offset and then to a function block 679 to generate a control value for the density of the aberrant engine and then to an End block 681.

[0117] In general, the average curve is loaded into the RIP such that the offsetting can be applied in a post-RIPPED job by the deviation that the print engine experienced from the average. It is noted that the useful linearization curve can be based upon a job-by-job basis. The reason for this is that there may be some situations wherein the user does not wish to utilize the linearization curve and, therefore, it can be stored with the job such that whenever the job is RIPPED, it will provide a stored RIP job with the linearization correction.

[0118] In the above noted calibration methods, it is important that there be a predetermined sequence to the operation. The first calibration that will be performed is that of calibrating the density levels in the color map blocks 618-622. This provides the color balancing. Thereafter, the density adjustments are made on the back of the engines with the above noted procedures of FIGURES 28a-28c. After these adjustments are made, the operation of the dot linearization is made. Since color dot linearization effects the color channels separately, some color shift may occur with linearization. After linearization, the correction for the halftone offset is made. Both operations are needed if additional color balancing to provide color matching is required.

[0119] Referring now to FIGURE 35, there is illustrated a block diagram of the software RIP described hereinabove. In general, the RIP is a conventional RIP which is defined by a block 670. The RIP 670 is a commercially available RIP which is utilized to rasterize received pages. The particular software RIP utilized in the present invention has the ability to perform the RIP operation depending upon the type of document that is received. If a contone image is received, it is RIPPED in one method, a color input is RIPPED in another method, a black and white job is RIPPED in another manner, as is a PostScript file and a PCL file. The RIP 670 has associated therewith on the input an input plug-in which is added at input plug-in 672, which is an added software interface. This interfaces with the input and allows

the software to define how the RIP 670 is to be configured. There are provided a multiplicity of channels 674 that define that configuration, each channel defining a particular configuration, *i.e.*, a contone configuration, a black and white configuration, etc. The RIP is then operable to provide the rasterized pages on an output 676 which interface with an output plug-in 678. The RIP 670, in addition to providing a rasterized image, also provides output information as to bit depth, resolution, number of pixels per line, etc. This information is provided to the output plug-in 678, which then forwards this information to a Print Station Manager (PSM) 680. This information is utilized to generate a page header for each page. This page header will define such things as the bit depth of the page, the page size, the number of colors, the number of planes, and the resolution, in addition to information regarding duplex and simplex printing. These are the characteristics of the particular page. This is then followed by the raw data which defines the bit values for the bit mapping image. The information associated with this raw data and the raw data are then stored and pointers placed in the Print Station Manager to allow this image to be later located. This image, of course, is also associated with job information, such that the page number of a given job is known. The Print Station Managers can utilize this to retrieve these jobs at a later time and manipulate them in any manner in order to determine how they will be printed, the order in which they will be printed, etc. After the Print Station Manager has received the information from the output plug-in 678, this information is then stored and later retrieved. A part of the PSM 680 is the electronic collator described hereinabove. This electronic collator is operable to generate virtual stacks for the document, which virtual stack correlates with the potential output configuration of the printers, which virtual stack will be directly mapped into that output configuration. For example, if there were four printers with four bins, each bin disposed over one on top of the other, the virtual stack would represent the stack that results in the four bins. In this manner, when the user pulls and stacks the contents of the bins, the stack will look identical to the virtual stack. This virtual stack is then divided up into job stacks, which are then buffered in a buffer area 682 for routing to print engines 684 under the control of the print engine objects (PPE) described hereinabove. A job manager 686 manages the operation of this. In general, the job manager 686 works on a job stack level,

whereas the PPE operates on a single page level. The PPE, in conjunction with the resource allocator, will look at the next instruction in the job stack and execute that instruction by fetching the particular page and printing that particular page. The PPE will also look at status information and define if an error exists, which will be relayed to the job manager 686. The job manager 686 can then process this error to reroute the remaining portion of the stack to a different printer or reroute the entire job stack to a different printer. Therefore, the job manager 686 monitors the error information and routes the stack to the appropriate engine, if necessary.

[0120] Referring now to FIGURE 36, there is illustrated a flowchart depicting the operation of the output plug-in. The flowchart is initiated at a flowchart 686 and then flows to a function block 688 to send the job information to the PSM. The program then flows to a function block 690 to retrieve the page information from the RIP, which is then sent to the PSM, as indicated by the function block 692. The program then flows to the function block 694 to retrieve a band of the raw data, this being the rasterized data. This band of data is essentially a small portion of the RIPPED page for all four color planes. This band of data is then compressed, as indicated by a function block 696. The purpose for performing the compression prior to forwarding to the PSM for storage is that this PSM 680 can be facilitated at a different location and compression facilitates throughput to a different location. After compression, the program will flow to a decision 698 to determine if any of the pixels in the different color planes are on. If so, the program will flow to a function block 700 along a "Y" path to set a boolean for the current color. This is essentially flagged and indicates that there is a color present in that particular plane. If it were determined that the color planes did not have any pixels turned on, this would indicate that this was a black-and-white job. If this were the case, the program would flow along the "N" path to the output of the function block 700 for all pixels in the plane. The lack of this boolean in the PSM would indicate that it was a black-and-white job and could therefore be processed in a more rapid manner. Once it is determined whether the boolean should be set or that there are no pixels on the color plane, the program will flow to a function block 702 to send the band data to the PSM, with the information regarding the boolean. The program will then flow to a

decision block 704 to determine if there are more bands. If so, the program will flow back to the input of function block 694 to retrieve another band of data and, if not, the program will flow to a function block 706 to send an End Page message to the PSM. The program will then flow to a decision block 708 to determine if the job is complete. If not, the program will flow back to the input of function block 690 to get the page information for the next page. When the job is done, the program will flow to the function block 710 to send an End of Job message to the PSM.

[0121]  In order to ascertain the toner level in each of the printer engines, the present invention utilizes a system whereby the actual rasterized image is evaluated as to the amount of toner that it will require. This is accumulated over a job stack which will be sent to a given printer. Once the job stack has been printed, an accumulation register is decremented indicating that the toner value has decreased for that particular print engine. Additionally, the accumulator can be examined prior to printing the job, but after rasterizing and determining the amount of toner that will be required, such that it can be determined whether there is sufficient toner available to run the print operation on that printer for that particular job stack.

[0122]  Referring now to FIGURE 37, there is illustrated a flowchart depicting the toner calculation operation. The flowchart is initiated at a block 720 and then proceeds to a decision block 722 to determine if a print job has been initiated. If not, the program will loop back around to the input. If so, the program will flow to the function block 724 to fetch a page of information for the given job and then to a function block 726 to accumulate values of pixels in that page. Each pixel will have a value that ranges from zero to 255 and these are summed up for the entire page. The program then flows to a function block 728 to divide the accumulated value by the total number of pixels in the page. This will provide the average value. If, for example, all the pixels were turned on to their maximum value, this average value would be 255. However, if only half of the pixels were turned on to full maximum value, the value would be 128. The program then flows to a function block 780 to define the value as a percentage of the black page. If it is 100 percent, that means all pixels are turned on to their maximum value. Any pixels that are turned off or have

a lower value will lower this percentage. The program then flows to a function block 772 to calculate the toner utilized for the given page, *i.e.*, this percentage will be multiplied by the maximum toner that will be required for that page for each color plane. It should be noted that in a color printer there are four toners, cyan, magenta, yellow and black. The toner usage for each plane will be calculated, such that if a given one of the toner cartridges for a given color is depleted, this can be indicated.

[0123] After the toner use for a given page is calculated, the program will flow to a decision block 780 to determine if the last page in the job has been processed. If not, the program will flow along a "N" path to a function block 782 to increment the job toner value, this being the total toner utilized for a given job. This will be divided up into the job stacks, as the job stacks are the smallest number of pages that will be sent to one of the multiple printers during a print job, the multiple printers defining the virtual printer. The program will then flow to the input of function block 724 to fetch the next page.

[0124] After the last page of the job, the program will flow along a "Y" path to a function block 784 to set the total toner value equal to the previous toner total value minus the job toner value, and the program will then flow to a decision block 786 to determine if the total toner value is less than a minimum value. If not, the program will flow to a function block 788, print the job, and then back to the input of decision block 722. If the total toner value is determined to have decreased below a minimum value, the program will flow to a function block 790 to send a toner low command to the user. The program will then flow to a function block 792 to reset the total toner value equal to the total toner value plus the job toner value, *i.e.*, the value before function block 784. The program will then flow to an end block 794. Along this path to the N block 794, the printer will be inhibited from printing that particular job stack. Once this error signal is returned to the job manager, the job manager can then generate an overall default or it can merely reroute the job to another one of the printers defined in the virtual printer with an error separator page provided thereby.

[0125] In general, the provision of the accumulated total toner value in a register provides, in effect, a "gas gauge" or "toner gauge" for a given toner module. This total toner value will be reset when a new toner is placed in the system. It will be reset to a value that approximates the known toner level. Additionally, characteristics of the toner module and the toner associated therewith in combination with the characteristics of the print engine will be utilized to calculate the amount of toner that is deposited on a sheet of paper with all pixels on to their maximum value. This is relatively easy to determine by running continuous pages through the printer until the toner has depleted to a value that is below an acceptable level. This will provide the total amount of toner output for a given number of pages, which will be proportional to that for a single page. This can then be divided down by the determined average value. This provides a significant advantage, in that it is an actual value, as opposed to an estimated value based upon the input print job. This is facilitated due to the fact that the rasterized image for each page in the job is collected prior to the job being printed. By comparison, conventional systems would have no method for determining usage of toner, since the rasterized image is developed in the printer after the processing section.

[0126] Referring now to FIGURE 38, there is illustrated a block diagram depicting a power on sequence, which is initiated at a block 796 and then proceeds to a decision block 798. Decision block 798 determines whether a power on condition has occurred, i.e., whether the system has been turned on. If not, the program will flow back to the input of decision block 790 and, if so, the program will flow to a function block 800 to set a variable "x" equal to "1." The program will then flow to a function block 802 to turn on engine "X" and then to decision block 804 to determine if a timeout has occurred. This program will loop back to the input of decision block 804 for approximately 10 to 15 seconds. At this time, the program will flow to a decision block 806 to determine if the value of "x" is equal to a maximum value. If not, the program flows to a function block 808 to increment the value of "x" and then back to the input of function block 802. When the value of "x" has reached its maximum value, the program will flow along the "Y" path to the input of decision block 798.

[0127]   The power-on sequence is utilized to minimize power requirements for multiple print engines with multiple fusers.  These fusers utilize heater lamps that have low resistance characteristics when the filaments are cold.  Therefore, when the filaments are initially powered, a current surge will result from a cold start.  As soon as the lamp filament temperature climbs appreciably, only several seconds or less, the current through the lamp stabilizes.  The initial current pulse is much greater than the average current at a given temperature.  Multiple (two or more) lamps on a single circuit could blow a breaker if the lamps are all started at the same time from a cold start.  By sequencing the power-on operation, the software can control each of the engines, since the print adapter allows control/status information to be transferred between the printer and the processing section.

[0128]   Referring now to FIGURE 39, there is illustrated a flowchart depicting the merge operation, which is initiated at a block 810 and then proceeds to a decision block 812 to determine if the merge operation is to be performed.  A merge operation, as described briefly hereinabove, is an operation wherein the Print Station Manager determines that two or more jobs are to be merged.  Since the pages have already been rasterized and stored, it is not necessary to go through the RIP operation again.  It is merely necessary to recreate a virtual job stack inserting the appropriate pages from one job into another job in the appropriate location.  Of course, it must be understood that these are rasterized images.  Of interest is the fact that these pages have associated therewith different printer characteristics.  For example, a 600x600 DPI job could be merged with a 600x1200 DPI job.  Since each page has associated therewith its own resolution, bit depth, page size, etc., the Print Station Manager need only send the job stack to a given printer with the PPE object then controlling the operation via the page.  Since the command information is associated with the page, each page can be treated differently by the PPE object and the print engine.  Therefore, a print engine could print one page at the 600x600 DPI level and the following page at the 600x1200 DPI level.  Even in the event that a color page were mixed with a black-and-white page, the color page would be sent to the color printer, followed by the black-and-white page sent to the same color printer

(although separate printers could be utilized with automatic finishing steps described hereinbelow), such that the color printer will first perform a color operation followed by a black-and-white operation.

[0129] Once the merge operation has been determined, the program will flow along a "Y" path to a function block 814 to receive the documents and pages to merge and the order thereof, this determined by the Print Station Manager. The program will then flow to a function block 816 to assemble the virtual job stack with the merge documents. This will create a hybrid job. The hybrid job and the associated virtual stack will then be divided into individual job stacks for the engines and then forwarded to the job manager and the printer, as indicated by function block 818. The program will then flow to the end block 819, as the remaining portion of the system in the form of the PPI objects and the kernel device drivers will take care of the printing at that time.

[0130] Referring now to FIGURE 40, there is illustrated a flowchart depicting the stack control, which is initiated at a block 820 and then flows to a function block 822. The function block 822 creates the virtual stack and then the program flows to a function block 824 to define the job stack for each of the engines. This essentially defines the borders between engines. The program then flows to a function block 826 to transfer the job stacks to the PPE object and then the program flows to a decision block 828 to determine if an error has occurred. This error is an error that has been returned by the engine, which is handled by the PPE object and then is relayed to the job manager. If the error occurs, the program will flow along a "Y" path to a function block 830 to create a separator page and then to a function block 832 in order to attach the separator page to the remaining portion of the error job stack, the error job stack being that portion left over after the generation of the error, including the page that did not get printed, if this is the case. The program will then flow to a function block 834, move the error job stack to another printer in the virtual printer set, such that the print operation can be completed. The program will then flow to a decision block 836 to determine if the print operation is done. The decision block 828, when an error has been determined not to have occurred, will also flow to

the input of decision block 836. If the job is not done, the program will flow along an "N" path back to the input of the error decision block 828 which will continue until all pages are printed. At this time, the program will flow from decision block 836 back to the input of function block 822.

[0131] It can be seen that this system automatically determines errors and, upon determination of an error, can automatically change the configuration of the system, such that a given job stack can be routed to a different printer. It is only necessary to create some type of separator page to indicate to an operator that an error has occurred and how the job is to be assembled.

[0132] Referring now to FIGURE 41, there is illustrated a flowchart depicting a page synchronization operation. Page synchronization is an operation whereby planes of color for each page are evaluated as to their completeness. When a complete plane has been printed, the next plane can be printed. If, for some reason, a given plane is not completely printed, it is possible that the system will go out of synchronization, such that the data for the next plane will be disposed behind that plane and will be sent to the FIFO in the print adapter. Therefore, it is necessary to know that all planes are being printed in a given print job, due to the speed of printing. Otherwise, a very large job could be destroyed.

[0133] The page synchronization operation of FIGURE 41 is initiated at function block 840 and then proceeds to a function block 842 to check the status signal received from the print adapter. The program then flows to a decision block 844 to determine if the page transmitted is complete. If not, the program will flow to a decision block 846 to determine if an error signal has been received, indicating that a problem has occurred and the page was not completely printed upon the generation of an EOP (End Of Plane) signal. If the EOP signal has not been received and an error signal has not been received, the program will flow to the input of function block 842 to again check the status. This will continue until an error is determined, at which time the program will flow from decision block 846 to a default block 848

and, when the page is complete with no error, the program will flow to a function block 850 to send a new page for processing.

[0134] Referring now to FIGURE 42, there is illustrated a flowchart for the operation in the print adapter. The print adapter is initiated at function block 852 to receive the page and then flows to a function block 854 to process this rasterized data through the FIFO. The program then flows to a decision block 856 to determine if an EOP signal has been generated. If not, the program will continue back to the input of function block 854. When the EOP signal has been generated, the program will flow along a "Y" path to a function block 858 to examine the contents of the FIFO. If the page has been completely printed, the FIFO 858 would be empty. This will be determined by decision block 860. If not empty, the program will flow to a function block 862 to generate an error signal and, if so, the program will flow to a return block 864 and transfer EOP signal out with no error.

[0135] Referring now to FIGURE 43, there is illustrated a block diagram of an alternate embodiment of the present invention. Documents are initially input to the software RIP as described above, represented by a block 866. The program then flows to a page store operation 868, which is operable to store the rasterized pages, which are then distributed by distributor 870, this being the operation described above with respect to the Print Station Manager and the job manager, in addition to the PPE objects. The distributor 870 is basically operable to develop the virtual stacks and the job stacks and configure them for routing to a multiplicity of print engines 872. The engines 872 are as described above and can be any type of engines, *i.e.*, color, black-and-white, etc. The distributor is controlled by the distributor control 876, which is operable to determine how the virtual stacks set in the virtual stack queue 874 are divided up into job stacks and routed to the various print engines 872. This is normally done as described above. The output of the print engines 872 are input to an automatic finishing device 878. The automatic finishing device 878 is a device that will automatically retrieve the contents of the print engines 872 and buffer them in a manner that they will automatically determine how the contents of the engines 872 are to be collated into the finished job output. Each

of the print engines 872 also provides status/control information on lines 880, which status/control information can return error information back to the distributor control 876. The distributor control 876 is operable to configure the automatic finishing device 878 and also reconfigure the automatic finishing device 878. For example, if the distributor control 876 had determined a particular routing configuration for the job stacks and one of the print engines 872 failed, the distribution control could redefine the job stacks and reconfigure the automatic finishing device 878. This, therefore, allows the system to run multiple jobs through by dividing them into job stacks and then treating each of the job stacks as individual entities and queuing them up and processing them independent of how fast another job stack in a given job is processed through an adjacent print engine. The automatic finishing device 878 will retrieve the output and place it in the proper order.

[0136] Referring now to FIGURE 44, there is illustrated a flowchart depicting the operation of the device of FIGURE 43. The program is initiated at a block 882 and then proceeds to a decision block 884. The decision block 884 determines if a new job is printed. If so, the program flows to a function block 884 to queue the instructions for the printer and then to a function block 886 to create a job separator with a finish code, this separator providing a code that can be read by the automatic finishing device 878. This could be as simple as a bar code. The program then flows to a function block 888 to execute the instructions in the job stack for a given printer and then to a function block 890 to fetch a given page and print it in accordance with the PPE object. The program then flows to a decision block 892 to determine if an error has occurred. If not, the program flows to a decision block 894 to determine if the instruction execution has been completed. If not, the program will flow back to the input of function block 890. If so, the program will flow to a decision block 896 to determine if the last instruction has been received. If not, the program flows to decision block 898 to fetch the next instruction and then back to the input of function block 888. When the last instruction has been received, the program flows along the "Y" path back to the input of decision block 884.

[0137] If a new job has not been received, the program will flow along a "node" path from decision block 884 around function blocks 885 and 886. Additionally, when an error has been defined, the program will flow from decision block 892 to a function block 900 to requeue the operation, *i.e.*, send it to a different print engine 872.

[0138] The automatic finishing machine 878 can be reconfigured by the distribution control 876 or, it can merely act in response to separator pages that are provided for each job stack. This will then merely require an individual manually moving the stacks from the print engine output bin to the automatic finishing device 878. Alternatively, the automatic finishing machine 878 could automatically extract the output from each of the print engines 872 and assemble it in accordance with the information received from the distribution control 876.

[0139] Although the preferred embodiment has been described in detail, it should be understood that various changes, substitutions and alterations can be made therein without departing from the spirit and scope of the invention as defined by the appended claims.